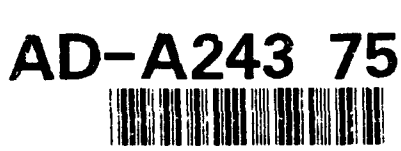


(1)

Frequency Domain Speech Coding

THESIS

Shane Switzer
Capt, USAF

AFIT/GCE/ENG/91D-10

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1991	Master's Thesis

4. TITLE AND SUBTITLE
Frequency Domain Speech Coding

5. FUNDING NUMBERS

6. AUTHOR(S)
Shane R. Switzer, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCE/ENG/91D-10

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Tim Anderson
AL/CFBA
Wright-Patterson AFB OH 45433

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The major goal of this research was to investigate speech coding techniques in an attempt to achieve high quality speech transmittable at 4800 bits per second. The approach taken to achieve this goal was to code the frequency domain representation of speech. Speech was represented by a sparse set of frequency components. Four frequency selections schemes were implemented, and the resulting frequency coefficients (magnitude and phase) were coded in an efficient manner for transmission. Specific techniques involved in the speech coder included: (1) a recurrent neural network architecture to make a periodic/noiselike decision, (2) the use of variable length windows for analysis and synthesis, and (3) a representation of noiselike speech using frequency banded energy information. The quality of the reconstructed speech was tested using listening tests which compared the different frequency selection schemes, along with original and sampled speech. The system did not achieve "toll quality" speech; however, the resulting speech was highly intelligible. Specific quality degradation was noted at window transitions.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Speech Coding, Speech Processing, Signal Processing	130
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

AFIT/GCE/ENG/91D-10

Frequency Domain Speech Coding

THESIS

Presentèd to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

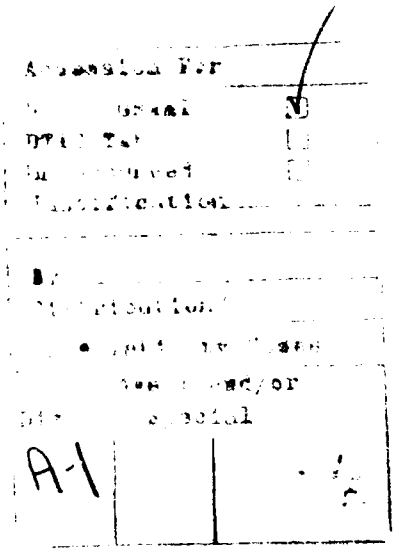
Requirements for the Degree of

Master of Science in Computer Engineering

Shane Switzer, B.S.E.E.

Capt, USAF

December, 12 1991



Approved for public release; distribution unlimited

Acknowledgments

When I began the process that lead to this document, I was under the impression that this would be *my* thesis. It was not very long before I began to realize that projects such as this are very much a group effort. Many of the ideas presented here came from individuals with much more experience and knowledge than I have. I would like to thank Captain Randy Lindsey for his suggestions and the use of his recurrent network software. I would also like to thank Dr. Tim Anderson of Armstrong Labs for providing the MRT tape, and arranging intelligibility testing at the Biocommunications and Bioacoustic Branch, as well as many suggestions for enhancements to the speech coder. I am grateful for the suggestions and direction provided by Dr. Bruce Suter and Major Steve Rogers. I would especially like to thank Dr. Matthew Kabrisky for his guidance and insight. Dr. Kabrisky provided constant challenges and encouragement throughout the entire thesis effort. I feel fortunate to have had the opportunity to work with such an enthusiastic and knowledgeable group of people.

It has been a very trying 18 months for me and my family. I would like to take time to mention my wife Gina, and my sons Nicholas and Michael. They were able to adjust to the long hours and lack of attention that AFIT creates. AFIT has renewed my belief in the importance of love and family. Without the support of Gina, Nicholas, and Michael, I surely would not have been able to complete this work.

Shane Switzer

Table of Contents

List of Figures

List of Tables

Abstract

The major goal of this research was to investigate speech coding techniques in an attempt to achieve high quality speech transmittable at 4800 bits per second. The approach taken to achieve this goal was to code the frequency domain representation of speech. Speech was represented by a sparse set of frequency components. Four frequency selections schemes were implemented, and the resulting frequency coefficients (magnitude and phase) were coded in an efficient manner for transmission. Specific techniques involved in the speech coder included: (1) a recurrent neural network architecture to make a periodic/noiselike decision, (2) the use of variable length windows for analysis and synthesis, and (3) a representation of noiselike speech using frequency banded energy information. The quality of the reconstructed speech was tested using listening tests which compared the different frequency selection schemes, along with original and sampled speech. The system did not achieve "toll quality" speech; however, the resulting speech was highly intelligible. Specific quality degradation was noted at window transitions.

Frequency Domain Speech Coding

I. Introduction

Background

Speech, the primary method of communication between humans, plays a great deal of importance in our daily lives. The telephone has become an integral part of our modern world, and the efficient transmission of speech has been studied extensively. Due to advances in technology, digital transmission of speech is now rivaling classic analog methods. Digital speech transmission offers advantages in both bandwidth and noise reduction. Systems have been developed that allow extremely low data rates for transmission of speech (in the 200 - 800 bits per second range). These low data rate speech coders produce speech which is intelligible, but low quality. No digital transmission methods have yet been developed that can transmit "toll quality" speech at 4800 bits per second.

In addition to speech transmission, other speech processing applications have arisen. Other applications include automatic speech and speaker recognition, enhancement of noisy speech, reconstruction of mutilated speech, and the development of devices for the hearing impaired.

The basis of development for several of theses systems is the known characteristics of speech signal itself. The system which produces speech, the vocal cords and the vocal tract, has been studied extensively, as have the spectral properties of speech signal itself. Not only has the transmitting apparatus (vocal system) been studied, but so has the receiving apparatus (auditory system). Knowledge of both systems is important to the design of speech processing systems.

Sundberg briefly describes the vocal system as follows:

> The voice organ is an instrument consisting of a power supply (the lungs), an oscillator (the vocal folds), and a resonator (the larynx, pharynx, and mouth). (30:82)

This nonlinear system generates the sounds that we know as speech. Sounds can be broken down into two general categories – voiced and unvoiced. Voiced sounds are produced when the vocal cords are vibrating. Vowel sounds are examples of voiced speech. Unvoiced sounds occur when the vocal cords remain separated (10:7). Sounds like "s" (as in sister), and "f" (as in for) are examples of unvoiced sounds. The system can be modelled, with some generality, as a linear time varying network of cascaded resonant circuits. Observed spectral properties of speech roughly determine the characteristics of this cascaded network (6:132). A further description of the system will be provided in the next chapter.

The auditory system also lends clues as to an efficient manner of representing speech signals. The human speech signal contains a large amount of redundancy. All information normally contained in the speech signal is not necessary for the transmission of a message. It is possible to completely understand speech that has been reproduced from only a handful of frequency components (1, 2, 14, 19). Some of this ability to extract meaning from a limited amount of information is probably due to the preprocessing accomplished by the auditory system (9). It has been shown that the auditory system acts like a set of parallel bandpass filters. The bandwidth of each of these filters is logarithmically scaled. Most of the information in the speech signal is carried in the lower frequencies. Due to the logarithmic scaling of the auditory system, lower frequency information tends to be enhanced. A brief description of the auditory system will be presented in the next chapter.

A system has been developed at AFIT that digitally encodes speech at 4800 bits per second. The reconstructed speech is clearly intelligible; however, it is not "toll quality." The primary complaint about the reproduced speech was that it seemed to contain a musical quality not present in natural speech. The system, developed by Capt Vance McMillan,

uses fixed length windows (40 milliseconds) to segment the incoming speech. A rule based system chooses which frequencies to represent the speech based on the spectral properties of the segmented speech signal. A decision is made as to what type of speech signal is present in the segment, (voiced, unvoiced, silence) and a different set of frequency selection rules is employed based on the type of speech detected. It is hypothesized that the rule based system has difficulties when a given segment of speech contains more than one type of speech (14:51). If one can determine the transition regions in speech, then McMillan's system could be modified to include variable length segments of speech (14:52). This could provide two advantages. First, the average window length could be longer than 40 milliseconds, thus allowing a lower data rate. Second, the musical noise that may be added by choosing incorrect frequencies to send for a given segment of speech may be eliminated. McMillan's system design is described in detail in his thesis, reference (14).

Problem Statement

The problem addressed by this research effort is the determination of transitions between voiced and unvoiced portions of speech, and the design a speech coding system that will include variable length segments of speech.

Assumptions

1. Capt McMillan's approach to encoding speech suffers problems that are due to the fact that a given segment of speech will contain more than one type of speech, and that segmenting the signal into variably sized analysis windows will overcome these problems (14:51–52).
2. The energy in the human speech signal decays at a rate of 6 dB per octave above 600 Hz, and has very little energy above 4 kHz (14, 18, 22). Since there is little energy above 4 kHz, sampling speech at 8 kHz will not introduce noticeable aliasing effects.

3. Human speech contains enough redundancy that choosing a small set of spectral components can provide an acceptable representation of the signal.

Scope

The vocabulary used to test the system is the set of words used in the Modified Rhyme Test (MRT). The MRT data consists of 272 words that rhyme in either the beginning or the end of the word. The words are contained in the carrier phrase 'you will mark *blank* please', where *blank* is the rhyming test word. A high quality reel-to-reel tape with the test words has been obtained from the Biocommunications and bioacoustic Branch of the Armstrong Laboratory.

Standards

By looking at the digital representation of speech, one can visually determine where transitions between different types of speech occur. The speech segmentation portion of the problem will be judged by comparing the results of transition region detection against a visual determination of the transition regions.

The speech compression system will be compared to McMillan's system (14). Additionally, it will be compared against a system designed by Capt Rick Ricart that uses a lateral inhibition network (LIN) to compress the results of windowed Fourier and wavelet analyses of a speech signal (22). Since mean squared error is not a good indicator of the quality of a reconstructed speech signal, subjective listening tests will be used. Recordings of compressed speech using these systems will be compared based on subjective quality tests similar to the one used by Capt McMillan. A detailed description of the subjective quality testing procedure will be outlined in Chapter IV.

In addition to comparing the quality of the reconstructed speech, it's intelligibility will also be tested. The intelligibility test will be performed by trained listeners from the Armstrong Laboratory at Wright-Patterson AFB. The intelligibility results will be compared with LPC-10 and the McCauley/Quatieri sinusoidal coder(19).

Equipment

All processing will be performed on the NeXT computer. The NeXT environment includes a Motorola 56001 Digital Signal Processor. The built-in DSP chip will be used for capturing, digitizing, and processing the speech signal. A built-in microphone jack is connected to an analog-to-digital convertor known as the CODEC (coder-decoder.) The CODEC samples the signal at 8012.8 Hz and quantizes it using an 8-bit μ-law quantization scheme.

Approach

Sampled, quantized speech will be converted from the μ-law representation to a linear quantization representation. Transitions between different types of speech will be determined. Once the transitions are determined, a standard windowed Fourier analysis will be performed on the given segment of speech. A decision will be made as to what type of speech is present for a segment of speech, and based on the decision, a representation of the speech will be generated. A method for selecting frequencies to represent the signal shall be determined. The selected frequencies will be encoded in a manner similar to that used by Capt McMillan. The encoded frequency information will be used to reconstruct the speech signal. Attention will be paid to the overall bit rate of the system, as well as the quality of the reconstructed signal.

Summary

This chapter described the overall problem addressed by this research effort. The problem includes locating transitions between differing types of speech, and using this information to design a frequency domain speech coding system. This chapter included the background, scope, assumptions, standards, and approach related to this problem. Chapter II discusses an approach to determining the transitions involving an artificial neural network. It also covers some of the known properties of speech, as well as briefly describes the human auditory system. Chapter III provides a more detailed description of the system

design. Chapter IV describes the testing method, and Chapter V discusses the results of the design and testing. Finally, Chapter VI provides conclusions and recommendations for future work.

II. Literature Review

Introduction

This chapter will provide a discussion of the background on which this thesis effort is based. It will begin with a discussion of coding systems developed at AFIT in 1990, by Capt Vance McMillan and Capt Rick Ricart. This will be followed by an introduction to artificial neural networks. A discussion of the the human vocal system will follow, and finally, an introduction to the human auditory system will be presented.

Background

Many speech coding systems have been developed based on the spectral properties of speech. These systems have provided a means to digitally transmit a speech signal. These systems are often quite robust (1, 2); however, they may require more bandwidth than analog transmission systems. McMillan has devised a system that encodes speech at 4800 bits per second (14:49). The resulting speech is clearly intelligible; however, it could not be called "toll quality" (14:50).

McMillan's speech coding system segments speech into intervals that are all of a fixed duration (40 milliseconds). A fast Fourier transform is performed on the segmented speech. Rules, based on the spectral properties of speech, are applied to the resulting frequency components. These rules pick a small set of frequencies to represent the given segment of speech. The rule based system tries to determine what type of speech is present in the segment of speech, and based on this decision it selects frequencies. It is quite likely that a given 40 millisecond segment of speech contains more than one type of speech. Most segments probably contain both voiced and unvoiced speech (8, 14). Since the rule base selects frequencies based on what type of speech is present, it will not send information about both types of speech in a given segment of speech. McMillan has hypothesized that if speech could be segmented where transitions take place, then a rule based system such as his could be used to produce "toll quality" speech at 4800 or fewer bits per second.

In addition to McMillan's system, Capt Ricart developed a system that would encode speech efficiently (22). Capt Ricart used several basis sets to decompose speech signals. He decomposed speech using the windowed Fourier transform, as well as the Gabor transform, and the Haar function as the basis for an affine wavelet transform. Capt Ricart's system used lateral inhibition networks to determine which coefficients to use to represent the original speech. Capt Ricart's did not attempt to quantize the coefficients, as his interest lay mostly in determining which type of decomposition provided the greatest compression ratio for speech.

Transition Detection Using Recurrent Neural Networks

Capt McMillan's system required a voiced/unvoiced/silence decision for each analysis window. In order to make the decision, the system thresholded on the energy contained within a window. High energy meant that the window contained voiced speech, low energy meant the window contained silence, and anything between the two thresholds was determined to be unvoiced speech.

Other methods for making classification decisions exist besides empirically determining thresholds. One of the most popular decision methods employed today is the use of artificial neural networks. Artificial neural networks have been applied at AFIT for image segmentation(27, 31), target identification(25), face recognition(5, 29), and speech recognition(28), as well as other applications.

Since the topic of this thesis is speech coding and since basic neural network concepts are throughly explained in other references, an in-depth background on neural networks is not provided. If terms like "supervised learning", "backpropagation", or "perceptron" do not sound familiar, the reader should to refer to an introductory article such as Lippmann (13) or to an introductory text such as Rogers and Kabrisky (24).

The basic way neural networks work is by taking a set of inputs, (known in pattern recognition terminology as *features*), and producing one or more outputs. Individual nodes process inputs in a manner inspired by neurons in living creatures. By arranging the nodes

of a neural network into different configurations, different types of processing can be accomplished. Artificial neural networks can be trained to distinguish between different classes of objects based on the input features. Given a set of features that can be used to distinguish between two classes of objects, and an appropriate architecture, a neural net can be trained to make the distinction between the classes for a given instance of an object.

A particular type of neural network architecture that has been studied recently is the recurrent neural network that uses the backpropagation algorithm for training. Lindsey (11:7) describes a recurrent network as follows:

> A recurrently connected neural network is a backpropagation network that contains feedback loops from previous states (timed inputs).

That is to say that previous outputs of the neural network can be fed back into the neural network as inputs. This process allows time dependence to be coded into weights of the neural network. A general structure of the recurrent neural network is shown in Figure 1.

Several algorithms to implement recurrent networks have been proposed. These algorithms include: backpropagation through time (BPTT), modified BPTT, real-time recurrent learning (RTRL), and subgrouped RTRL. Lindsey, in his thesis(11), details these algorithms, as well as adds enhancements to the RTRL algorithm.

The strength of recurrent networks lies in their ability to encode and learn time dependent processes. Rogers proposed that since speech characteristics change with time, recurrent networks may be useful in determining transitions between voiced and unvoiced portions of speech (23). Recurrent networks allow a decision based not only on the current input, but also upon the previous network output. Depending on the window size, there must be some correlation between whether the next portion of speech is of the same type as the present portion of speech. This is the type of problem that recurrent networks should be able to learn (12).

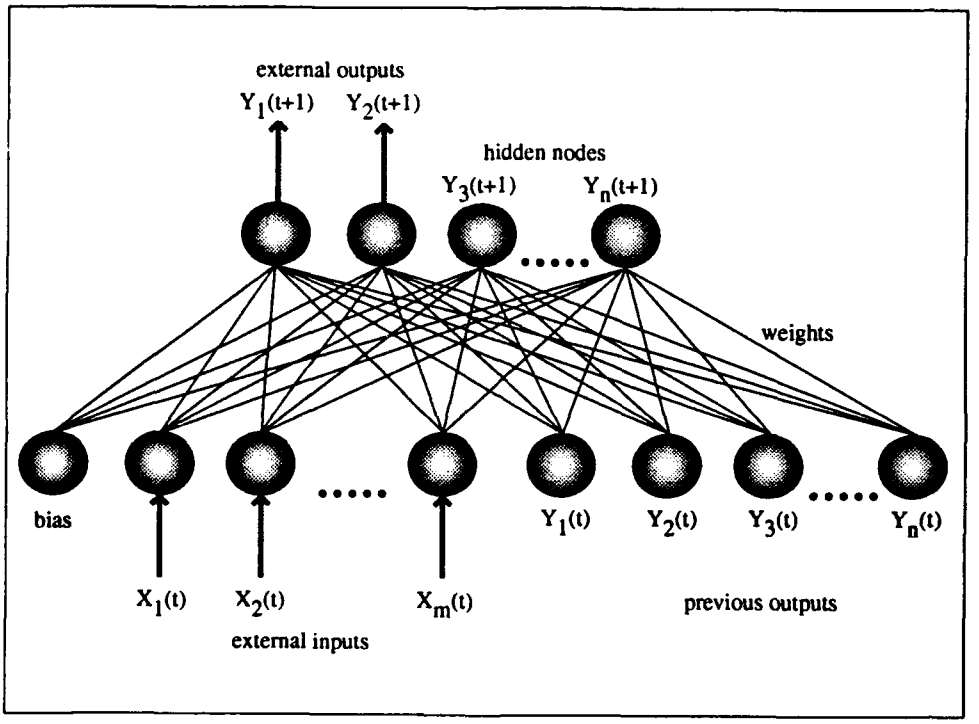


Figure 1. General Recurrent Neural Network Architecture - notice the feedback loop, previous outputs become inputs. From Lindsey (11).

This section has briefly discussed the use of neural networks as decision structures. It provided the general architecture for a recurrent neural network, and motivated the use of this type of network as a tool for transition detection in speech. The next section discusses speech generation and the spectral properties of speech.

Speech

Speech Organs. The organs of speech are broken down into three categories:

1. Lungs and Trachea
2. Larynx
3. Vocal Tract

The lungs and trachea form the power supply that is used to generate speech. The diaphragm and abdominal muscles contract the lungs and force air through the trachea. The trachea is a tube that connects the lungs to the larynx (18:59–62).

The larynx consists of the muscles and cartilage. Together, these muscles and cartilage form the vocal cords. The vocal cords are folds of flesh that are stretched across the larynx. The opening between these folds is known as the *glottis*. It is this part of the vocal system that causes the oscillatory nature of voiced speech (18:62).

The vocal tract is considered to be any part of the speech organ located above the vocal cords. The vocal tract consists of the remainder of the larynx, the pharynx, the mouth, and the nose. Certain structures within the vocal tract, particularly the lips and tongue, play an important role in the production of speech. The shape of the vocal tract determines resonances in speech. The resonances are the distinguishing features of the different vowel sounds (18). The vocal tract and larynx are depicted in Figure 2.

Speech Production. Speech is divided into excitation and modulation. The vocal tract can be excited in several ways. Two broad categories of excitation exist – voiced and unvoiced.

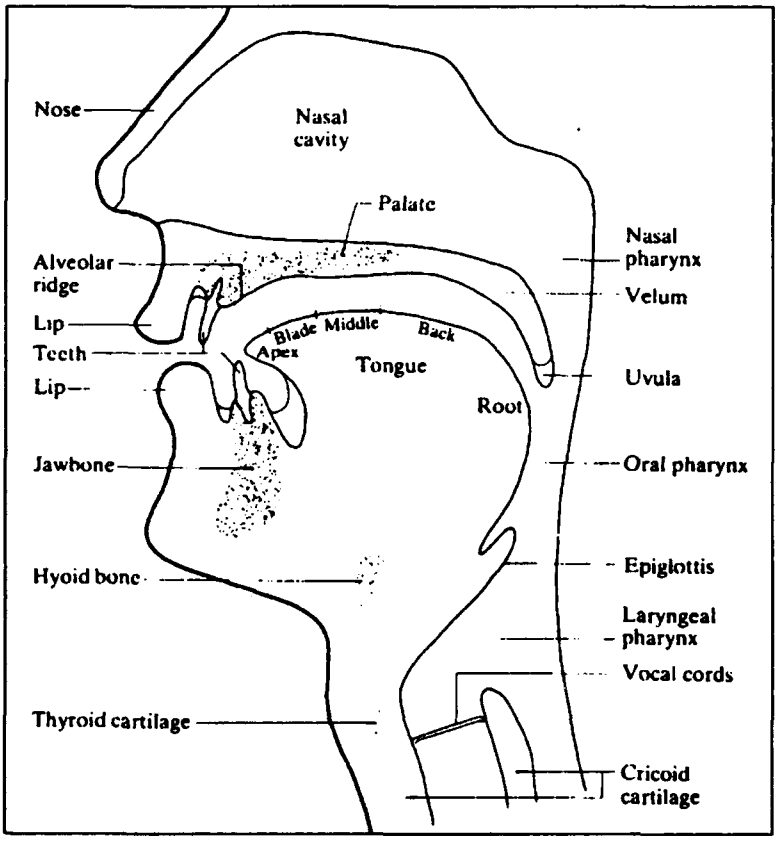


Figure 2. The Vocal Tract and Larynx. From Parsons (18).

Excitation. Voiced excitation (*phonation*) is the most important excitation source and is caused by the oscillations of the vocal cords. These oscillations are the result of the vocal cords being stretched together, while air is being forced out of the lungs. The stretching of the vocal cords causes the glottis to close. Pressure builds up on the lung side of the larynx and forces the vocal cords apart, thus opening the glottis. The air now rushes past the glottis, and Bernoulli effect forces the glottis closed again. Once again, air pressure builds up and the cycle is repeated. This periodic opening and closing of the glottis creates the oscillatory excitation (18:64–66). The rate of oscillation is known as the *fundamental frequency*, *glottal frequency*, or *pitch*. The average adult male glottal frequency ranges from 80 to 160 Hz. For females this range is from 160 to 400 Hz (18:98). Along with the glottal frequency, several higher harmonics are also formed. (30:82)

Unvoiced excitation of the vocal tract is created when air is forced through a constricted passageway in the vocal tract. Unvoiced excitation is nonperiodic in nature, and is viewed as wideband or broadband noise. The type of noise is dependent upon where in the vocal tract the constricted passageway is located. Phonated excitation can be passed through a constricted opening and unvoiced excitation will result. (18:65–66).

Modulation. Modulation is how humans change the vocal tract to produce different sounds, for both voiced and unvoiced speech. Changing the shape of the vocal tract causes different resonances to occur. The vocal tract can be considered to be a horn or tube. The length and the diameter of the horn can be varied by the speaker.

Waves travel through tubes according to the wave equation. Sinusoidal solutions of the wave equation produce a set of standing waves that match the boundary conditions of the tube. Resonances occur at frequencies associated with these standing waves. In speech, these resonant frequencies are called *formant* frequencies (or *formants*). For a typical adult male, the distance from the glottis to the lips is about 17.5 centimeters (30:82–83). A perfect cylinder of this length, closed on one end, would have its first four resonances at about 500, 1500, 2000, and 3500 Hz. These resonances would correspond

to the first four formants (30:82–83). Changing the shape of the vocal tract affects the frequency of these formants. Formants are the basis for the differences heard between vowel sounds. Figure 3 provides a model for voiced speech.

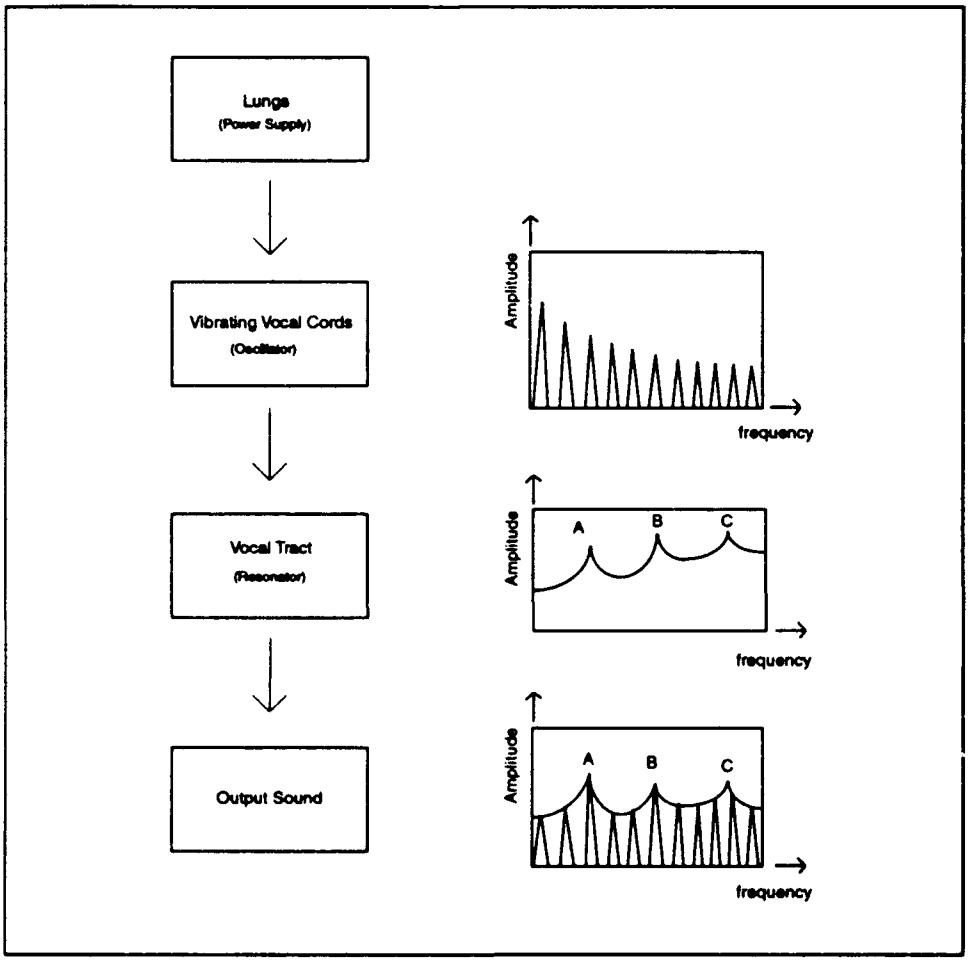


Figure 3. Voiced Speech Production Model. Adapted from Parsons (18).

The speech generating system is often modelled as an electrical circuit consisting of a pulse generator and a linear network of cascaded resonant circuits (18:124–125). For voiced sounds, the pulse generator emits pulses at a periodic rate. For unvoiced sounds, the pulse generator emits broadband noise. The resonating circuit is modelled as a set of overlapping bandpass filters. The centers of these bandpass filters correspond to

formant frequencies. Pulses pass through the filters and the resulting output is the electrical equivalent of speech. This model is the basis for linear predictive coding (LPC) of speech, as well as other speech coding techniques (18:124–125).

For a more complete introduction to speech, see Parsons (18). The next section will provide an introduction to the other important part of speech communication – the auditory system.

The Auditory System

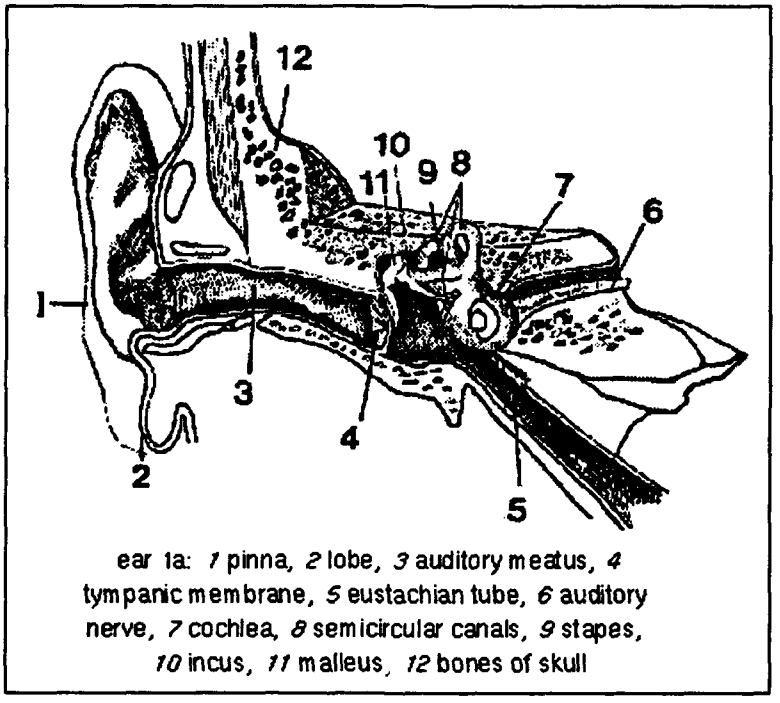
At this point in time, the perception of hearing is not well understood. This section will cover the preprocessing of sound believed to be performed by the ear. It will not detail any of the high level processing performed by the brain. This material is intended to be an introduction only, and is by no means complete.

The ear is divided into three sections:

1. The outer ear
2. The middle ear
3. The inner ear

Figure 4 provides a view of the structure of the ear. The outer ear consists of the *pinna* (visible, funnel shaped cartilage), the *meatus* (external canal), and the *tympanic membrane* (ear drum). The middle ear is an air filled canal which contains the *ossicular system*. The ossicular system is a chain of three bones (*ossicles*). These bones are the *malleus* (hammer), the *incus* (anvil), and the *stapes* (stirrup). The inner ear is a shell shaped structure known as the *cochlea*. The entire system acts as a transducer; converting the energy of sound waves into electrical impulses that are sent on to the brain (18).

The transformation from sound waves to electrical impulses begins at the tympanic membrane. Sound waves strike the membrane and cause it to vibrate. These vibrations are transferred through the malleus, the incus, and stapes to the cochlea. This system



ear 1a: *1* pinna, *2* lobe, *3* auditory meatus, *4* tympanic membrane, *5* eustachian tube, *6* auditory nerve, *7* cochlea, *8* semicircular canals, *9* stapes, *10* incus, *11* malleus, *12* bones of skull

Figure 4. The Ear. From Webster (16).

provides impedance matching between the sound waves in the air and the fluid waves inside the cochlea (7:570–571). Besides providing impedance matching, the ossicular system protects the auditory system from loud sounds. Muscles connected to the stapes and malleus contract 40 - 80 milliseconds after a loud sound is received by the auditory system. The contraction of the muscles stiffens the ossicular system and attenuates low frequency sounds. This attenuation reflex probably also helps mask low frequency sounds, thus acting as a filter against background noise, thus allowing a person to concentrate on sounds above 1 KHz (7:570–571).

The cochlea is a coiled system of tubes. The diameter of the cochlea decreases as one moves from its base to its apex. The cochlea consists of two main chambers which are separated at the basilar membrane. The faceplate of the stapes is attached to the oval window at the base of the cochlea. Vibrations in the stapes cause waves in the fluid of the cochlea, which in turn, cause the basilar membrane to vibrate. Near the base of the cochlea, the fibers of the basilar membrane are short and stiff. Near the apex the fibers are longer and more limber. The short stiff fibers vibrate at higher frequencies, and the longer more limber fibers vibrate at lower frequencies (7:571–572).

The cochlea also contains hair cells that when displaced cause neurons to fire and transfer incoming sounds to the brain. These hair cells are displaced as the basilar membrane is displaced (7:573–574). Different frequencies cause displacements of the basilar membrane at different places along its length. Since hair cells are displaced by motion of the basilar membrane in the same vicinity, it is believed that different hair cells are displaced according to the frequency of incoming sounds. The resulting system acts as a mechanical-neural spectrum analyzer (18:69). This spatio-frequency model of cochlear processing is known as the *place theory* (7:573,575).

Using the place theory for frequency perception, the frequency response of the basilar membrane is a function of the distance from the stapes. Higher frequencies ring near the stapes, and lower frequencies ring near the apex of the cochlea. The frequency response can be considered to be a series of overlapping bandpass filters. The bandwidths

of theses filters are not known precisely; however, it is known that these bandwidths increase logarithmically with frequency. These logarithmically increasing bandwidths produce a bias in the auditory system towards low frequency input. In this way, the auditory system seems well adapted to processing speech signals (8). Figure 5 shows the frequency response of the cochlea as a function of distance from the cochlea.

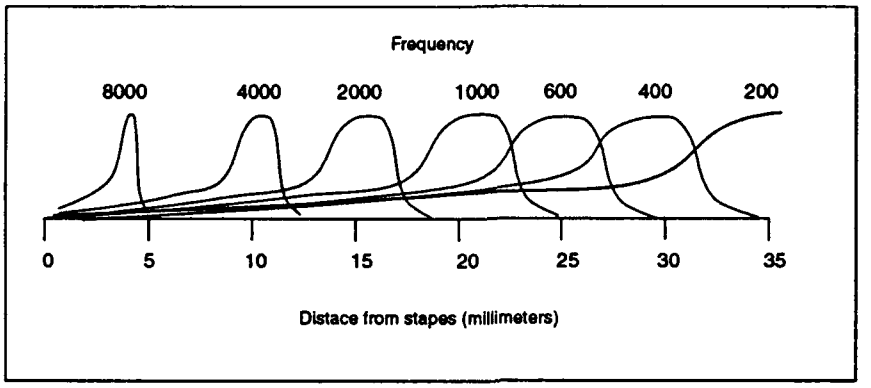


Figure 5. Frequency Response of the Basilar Membrane. Adapted from Guyton (7).

Evidence suggests another possible theory of frequency resolution. Up to about 5 KHz, the auditory nerves fire at the same rate as the incoming sound. Researchers believe that the brain may use this information to determine the frequency of incoming sounds (7:575) (18:69–70). Others argue that if frequency information is obtained by this method, then what is the purpose of the elegant cochlear system?

This brief introduction to the auditory system has been highly simplified. Feedback from the brain plays an important role in the process of hearing, but is not yet well understood. The purpose of this discussion was to introduce a model for some of the preprocessing performed by the auditory system, and to motivate a frequency selection scheme for the speech processing system.

Summary

This chapter has introduced the concept of artificial neural networks. The neural network approach has been used in several contexts, including decision making and function prediction. Neural networks may prove useful in transition detection. Speech production was addressed, both from the standpoint of the mechanics of the system, and also from the spectral properties of the speech signal itself. A brief overview of the auditory system was also provided.

The next chapter covers the design of a system to segment speech into voiced and unvoiced regions. A rule based system is used to select frequencies to represent a given segment of speech. The system will reconstruct speech from the reduced set of frequency components.

III. System Design

Introduction

The system simulates a low data rate speech coder. The system is based on the frequency domain representation of speech. Since speech tends to be periodic, the frequency domain is a natural place in which to process speech signals. Much research has been directed toward coding speech in the frequency domain, and this thesis attempts to extend some of the work previously done here at AFIT (1, 2, 14, 22). The basic idea pursued in this research is to try and reproduce the speech signal as accurately as possible (in a perceptive sense) using it's frequency domain representation. Since the time domain representation of the signal can be completely recovered if all of the frequency domain information is provided, the question is: "How much of the frequency domain information can be lost without seriously affecting the intelligibility and quality of the reconstructed signal?". The design of this system attempts to remove enough of the frequency domain information to achieve a data rate of 4800 bits per second, while maintaining a high level of quality and intelligibility.

Processing Environment

Work done on this project has been performed on the NeXT computer environment. NeXT computers provide a solid, basic foundation for digital processing of speech. NeXT computers come with a built-in Motorola 56001 Digital Signal Processor (the DSP). This built-in DSP chip is used to capture, digitize, process, and play back the speech signal. The NeXT also includes a microphone jack which is connected to the analog-to-digital convertor known as the CODEC (coder-decoder), a headphone jack, and dual stereo output jacks. All FFT's are performed on the DSP chip. Other numerical processing is performed by the NeXT's Motorola 68040 in 32 bit floating point format.

Software is primarily written in ANSI C with some C Macros and Objective C included. NeXT specific function calls were employed to handle the speech objects and DSP operations. The NeXT system considers sounds to be objects that can be manipulated through object-oriented message passing. The sounds are stored in C structures and can also be manipulated using standard C techniques. Since I was working on a prototypical system with no specific implementation in mind, no attempt to treat the system in an object-oriented manner was considered. I tried to keep the data structures, as well as the programming techniques straight forward.

The System

Figure 6 provides a block diagram of the system. Explanations covering each block of the system are provided in the following paragraphs.

Preprocessing Analog speech is sampled via the CODEC at 8012.8 KHz. For calculations and analysis, the sampling frequency is considered to be 8 KHz. The CODEC stores the sampled data using an 8-bit μ-law format. The 8-bit μ-law format provides a dynamic range approximately equal to a 12-bit linear format (4). Using the μ-law format does introduce quantization noise which is different from the noise of a linear quantization scheme. Rabiner and Schafer (21:188-195) provide a discussion of the noise added using a μ-law encoding scheme. The 8-bit μ-law digitized speech is converted to a 16-bit linear format for subsequent processing. This conversion is performed by some NeXT specific C function calls. Some error is introduced by this process; however, there is no perceptible difference in the playback of sounds recorded in 8-bit μ-law format, and playback of those which have been converted to 16-bit linear format.

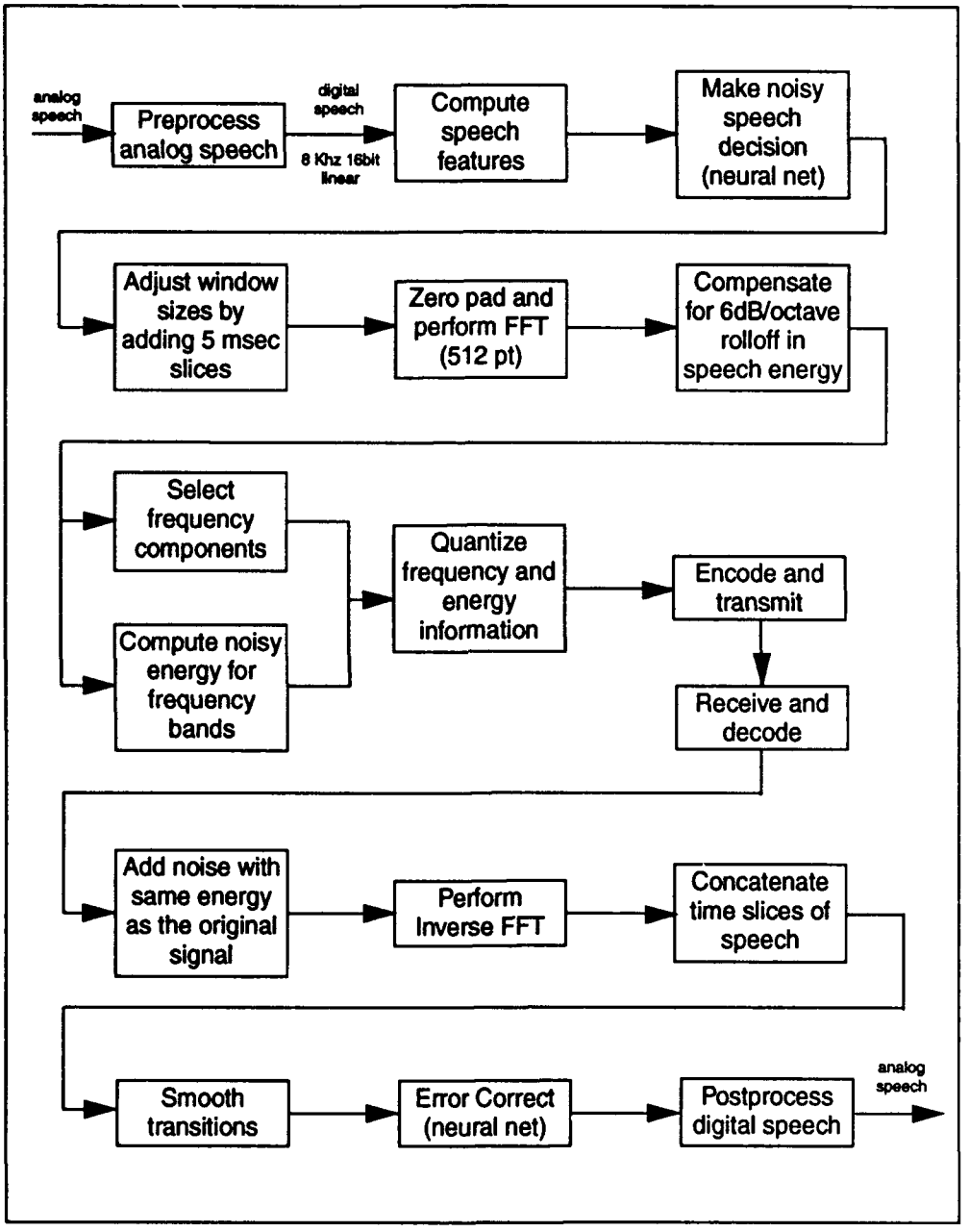


Figure 6. Block Diagram of the Speech Coding System

The 16-bit linear formatted signal is stored internally as integers. In order to use the DSP chip to perform FFT's, it is necessary to convert the signal from integers to floating point numbers, and to scale the floating point numbers to values in the range of [- 1.0,+1.0]. This is accomplished by dividing each of the sample values by the largest magnitude of all of the samples in the sequence. The digital speech sampled at 8 KHz, is now a sequence of floating point numbers between -1.0 and 1.0, and is ready for processing by the speech coding system.

Computing Speech Features The system treats different portions of speech differently. For example, if a portion of the speech contains a fricative, then the spectrum of the speech signal looks like the spectrum of a noisy signal, and the system treats the signal as if it's noise characteristics are important. If a portion of speech is made up of a vowel or vowel-like sound, then the spectrum of the speech looks like the spectrum of a periodic function, and the system treats the speech as though the dominant frequencies are important. The key to how the system will treat a particular portion of speech, depends upon the characteristics of the speech signal itself. Since the system must know how to treat a particular portion of speech, certain characteristics (or *features*, to borrow from the pattern recognition vernacular) must be obtained to base a periodic/noiselike decision. It should be noted that the decision is not a voiced/unvoiced or a fricative/vowel decision, but rather a periodic/noiselike decision. Although voiced or vowel sounds will often fall into the periodic category, voiced fricatives and stops will be noiselike, but will also have underlying voicing characteristics. The degree to which a portion of speech seems noiselike is important to subsequent processing.

Common features used to base a voiced/unvoiced decision are energy and number of zero crossings per time slice of speech. These features are also typically used to determine speech versus silence (21:130-134). These features are easily obtained by processing speech in the time domain.

In addition to the above mentioned features, two other features are computed to base a periodic/noiselike decision on. They are: 1) the number of sign changes of the slope between consecutive samples, and 2) the ratio of energy in frequencies below 500 Hz to the total energy of a time slice. Noiselike speech will tend to have more sign changes in the slope between two consecutive samples, and periodic speech will tend to have more energy concentrated in the lower frequencies. For voiced fricatives, there may be fewer zero crossings than slope changes. Thus, including slope changes as a feature may allow the neural network to distinguish voiced fricatives as noiselike speech. The number of slope sign changes is easily computed in the time domain, while the low frequency energy to total energy is computed in the frequency domain.

Periodic/Noiselike Decision The features just described are used to make the periodic/noiselike decision. The typical method for making such a decision is to compute the features for several utterances, and set a threshold based on empirical results. Statistical analyses of the features may also be used to set a threshold for such a decision. These methods can be time consuming and somewhat arbitrary. Neural networks offer an interesting alternative to these typical methods of decision making.

The speech coding system uses a neural network architecture to determine if a particular portion of speech is noiselike or periodic. Neural networks are briefly explained in Chapter 2. For a good introduction to neural networks, see Rogers and Kabrisky (24). Ruck (26) has shown that neural networks trained using the backpropagation algorithm approximate the Bayesian Optimal Discriminant function . That is, based on the inputs of a particular set of features, a neural network, if trained properly, will produce the decision with the highest probability of being correct. The big advantage to using a neural network is that one only needs to provide the feature values and a correct output, and through the backpropagation algorithm, the network will learn the statistics of the inputs to correctly classify the candidate output that would most likely have the given input feature values.

The particular type of network used for the periodic/noiselike decision is a recurrent neural network trained with backpropagation. For a more detailed discussion of recurrent network architecture and the training algorithm used in this system, see Lindsey (11). The architecture of this particular net is provided in Figure 7. The network has six inputs which will be further described below. The network operates by performing a sigmoidal $1/(1+e^{-x})$ function on the weighted sum of it's inputs (features).

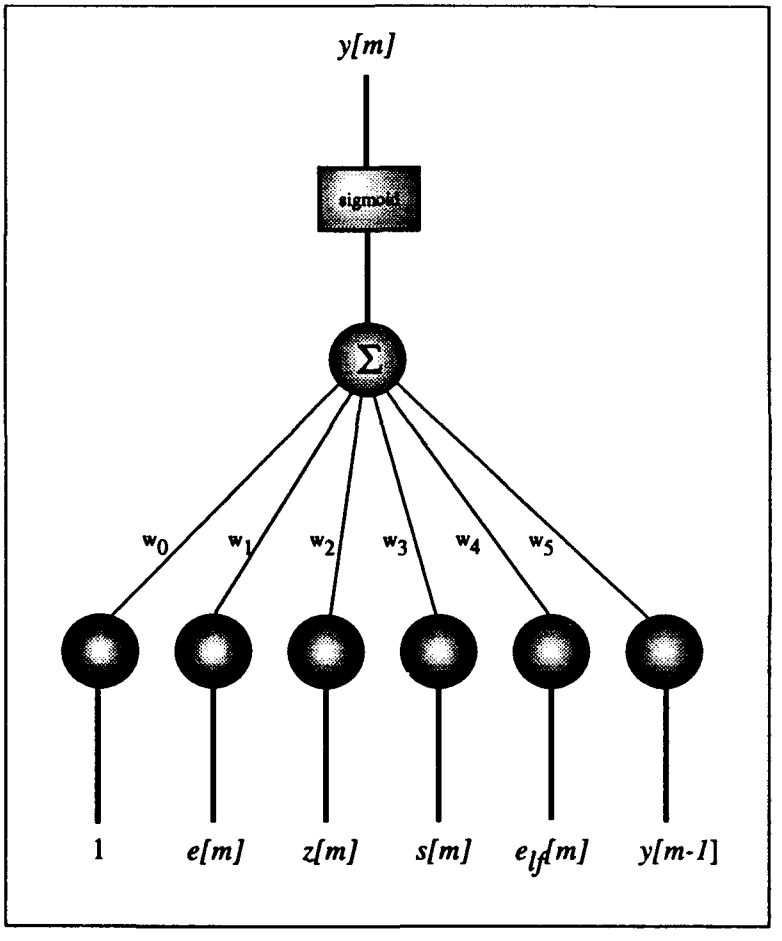


Figure 7. Noiselike Decision Recurrent Neural Network.

Based on the output of the function, a periodic/noiselike decision is made. The output $y[m]$ for a current set of features is defined as

$$y[m] = \frac{1}{1 + e^{-(w_0 + w_1 e[m] + w_2 z[m] + w_3 s[m] + w_4 e_{lf}[m] + w_5 y[m-1])}} \quad (1)$$

where $e[m]$ is equal to the energy in time slice m, $z[m]$ is equal to the average number of zero crossings in time slice m, $s[m]$ is equal to the average number of slope sign changes in time slice m, $e_{lf}[m]$ is the ratio of energy below 500 Hz to the total energy in time slice m, and $y[m-1]$ is the previous output of the network. Each time slice m represents 32 samples of the digital speech signal. With a sampling frequency of 8 KHz, each time slice is 4 milliseconds long. The values $e[m]$, $z[m]$, $s[m]$, $e_{lf}[m]$, and $y[m-1]$ make up an input vector. The weights w_0,w_1,w_2, w_3,w_4, and w_5 were determined by training the network.

Training is accomplished using a backpropagation algorithm. The backpropagation algorithm works by adjusting the weights of the network based on the error between the desired output and the actual output of the sigmoidal function. Large errors force the weights to be adjusted more than small errors, and training rates also affect how much the weights are adjusted. When training is complete, (i.e. the user is satisfied that the network correctly identifies a certain percentage of training input vectors or test input vectors correctly) the weights can be fixed and the resulting network can be implemented in either hardware or software.

The software used to perform the training was provided by Captain Randy Lindsey. For his thesis (11), Lindsey created a software implementation of the recurrent neural network which included a variable learning rate. For further details of recurrent networks, see Lindsey (11).

This particular network was trained on three separate utterances by the same speaker. The network correctly classified 98.4 percent of the training set. The weights were then fixed and used to process the rest of the utterances that are used for testing the system. If this type of decision network were to be used in an actual system, several more training vectors from several different speakers would be necessary. Figure 8 shows a plot of the decision made by the recurrent network. The plot displays the speech signal along with the decision made by the network. A noisy decision region is plotted as a 1 on the plot, and a non-noisy decision region is plotted as a 0. The actual output of the sigmoidal node of the recurrent network is close to 0 or 1. The plot shows results rounded to the nearest whole number.

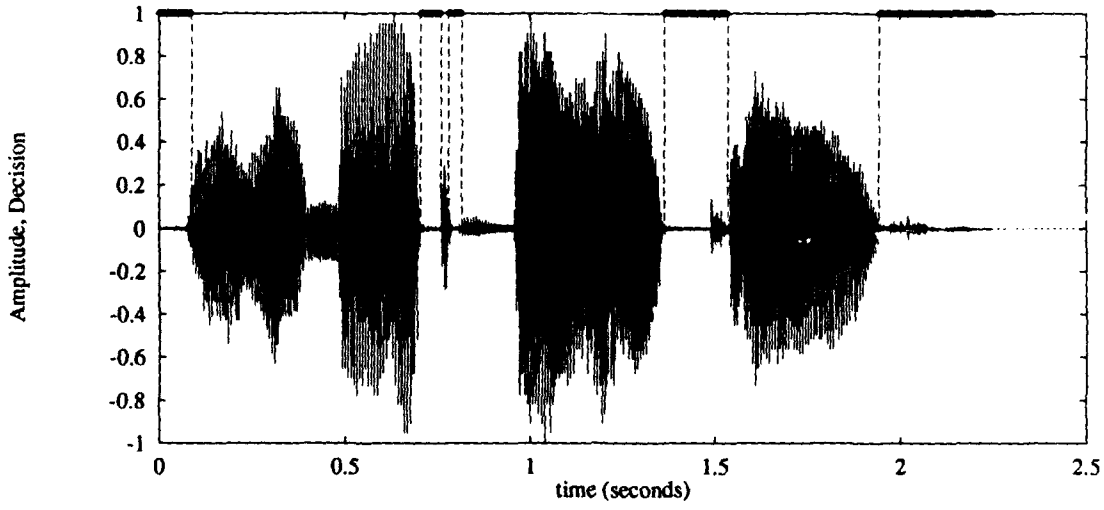


Figure 8. Noisy Decision Plot - Noisy regions plotted as 1, non-noisy plotted as 0.

One advantage of the recurrent type of network is that it's output depends on it's current inputs plus its previous output. Since speech tends to remain stationary for periods of 30 to 40 milliseconds or longer (14:20), it is to the net's advantage to know what type of speech it just processed. Recurrent networks are often used in function prediction. It is speculated that if there is an underlying function which describes a signal, then recurrent networks can often learn the function and predict the value of the function some time in the future. If there is some sort of function which underlies the periodic/noiselike attribute of continuous speech, then perhaps the recurrent net can determine it.

Recurrent networks are very new, and little is understood about their strengths and limitations. They do provide a novel and useful approach to the periodic/noiselike decision process. Although this network seems to perform well, other recurrent network architectures may do a better job of correctly classifying the inputs. Since it is hard to know the dynamics of how the network learns or remembers the underlying function, questions of optimization are difficult to answer.

In addition to this type of network architecture, a multilayer perceptron, using backpropagation was also tested as a candidate for making the periodic/noiselike decision. The multilayer perceptron did not fare as well as the recurrent net in correctly classifying the input. Again, questions as to the best architecture, features, and training inputs for the decision process do not have simple answers.

Analysis Window Size A problem with most speech coding systems is that a fixed window size is employed. Since coding systems often process speech based on content (voiced/unvoiced, periodic/noiselike), it is often necessary to treat an entire window as if it were all the same type of speech. If a transition between one type of speech and another occurs within a window, systems may be forced to treat voiced speech as unvoiced speech or vice-versa. The resulting reconstruction will likely have unacceptable errors in these windows.

One solution to the problem of using fixed window sizes, is to make the windows so small that each window will contain only one type of speech. Transition regions in speech will then have their own small window, and errors produced by windows with different types of speech will be infrequent and will not persist as long as they would if a longer window is used. Small windows may eliminate the errors associated with treating a larger window with a mixture of speech as one type or another; however, if parameters must be sent for each window, then fewer parameters are available for smaller windows. For example, if for 40 milliseconds of speech, one can send 16 parameters, then for a 10 millisecond slice of speech, one can only send 4 parameters. Sixteen parameters may be sufficient to code a portion of speech; but, 4 parameters may not be. There may also be some fixed overhead required for each set of parameters, so that shorter windows will be required to send these bits more frequently than longer windows.

This system attempts to overcome some of the problems with fixed window sizes by using variably sized windows. It is assumed that as long as the speech remains relatively stationary over the entire window, then longer windows will allow more parameters to be sent and a better representation of the signal will be possible. Short windows are used to handle portions of speech in which the characteristics of speech are changing rapidly. The short windows are most often used for the release portion of the plosives, and to handle the transition regions between periodic and noiselike sounds. Longer windows handle

the longer lasting vowel and vowel-like sounds, as well as silence. More parameters can be used to code the longer windows, hence the quality of speech processed using longer windows should be better. Short windows are usually treated as noiselike sounds. The details of how noiselike sounds and periodic sounds are treated will be covered later. It should be noted though, that the system places importance on the energy in noiselike sounds, whereas it places importance on frequency content for periodic sounds.

The smallest window size used by the system is 128 samples or 16 milliseconds of speech; the longest window is 512 samples or 64 milliseconds of speech. Windows can increase in size from 128 samples to 512 samples by increments of 32 samples. The neural network decision structure determines the type of speech (periodic/noiselike) for the base window size (128 samples). The base window size is extended if the next 32 samples of speech are of the same type as the base window. As long as the type of speech remains the same, the window gets extended until it's size reaches 512 samples. When the neural network finds that the type of speech is changing or the window size reaches 512 samples, the window size becomes fixed. This window size is used throughout analysis and synthesis for the particular portion of speech. The window size will also determine the number of spectral components that can be transmitted for the particular portion of speech. With the analysis window size set, the system can now convert the signal to the frequency domain for further processing.

Transforming the Speech Signal to the Frequency Domain In order to process the given analysis window of speech, it must be transformed from the time domain into the frequency domain. This is done by performing an FFT on the signal. For efficiency, all FFT's are performed by the DSP chip. The FFT size used in 512 points. Analysis windows may be shorter than 512 samples, so zero-padding may be necessary. Using a 512 point DFT on data that is sampled at 8 KHz, provides a frequency resolution of 15.625 Hz per frequency bin. The system uses rectangular windows on the input speech data. Rectangular windows have larger side lobes than do other windows, such as the

Hamming or raised cosine windows. Rectangular windows do have the added advantage that the *overlap and add* method is not required for analysis and synthesis of the speech signal. Window overlap of 50 percent is typical in other analysis/synthesis systems. Since each portion of speech is thus analyzed and synthesized twice, the number of parameters that can be sent per window is cut in half. Rectangular windows are also convenient when variable analysis window sizes are used. One problem with rectangular windows is that discontinuities at window borders can lead to a "popping" noise being introduced in the reconstructed signal. The overlap and add method for reconstruction used for other windowing schemes does not suffer from this problem. A smoothing process which acts on the time domain reconstruction of the signal reduces these "pops". A description of the smoothing process is described later in this chapter. Once a frequency domain version of the signal is available, the system is ready to select which frequencies it shall use to represent the signal.

Frequency Selection When choosing which frequencies to transmit to represent the signal, several possibilities are available. One can choose the frequencies that will provide the least mean squared error between the original signal and it's reconstruction. Least mean squared error may not, however, provide for the best sounding reconstruction. The least mean squared error is obtained by selected the frequency components that have the largest magnitudes. Picking frequencies that have the largest magnitudes makes the system behave like a low-pass filter. This is due to the fact that energy in the speech signal tends to roll off at about 6 dB per octave for frequencies above 600 Hz. Since higher frequencies have less energy, they tend to get ignored when picking the maximum amplitude frequencies. Could it be possible that the higher frequencies are important in the perception of speech sounds? Observing the spectrum of the windowed speech signal shows that most peaks also have neighbors that have large amplitudes. As long as a peak gets picked, is it important to the perception of hearing that it's neighbors also get picked, or are the magnitude of the neighboring frequencies a function of the windowing process?

Alenquer(1), Bashir(2), and McMillan (14) all used a method in which an estimate of the glottal pitch was made, and harmonically related frequencies were selected. They all achieved some measure of success with their selection schemes; however, different approaches to frequency selection were tested in this research. Four frequency selection schemes were used in an attempt to find a method for frequency selection that would provide the best quality reconstruction from an auditory perception point of view. The following paragraphs discuss each of the four selection schemes.

The first scheme was a simple max-picking method. After taking the FFT of a time slice of speech, the M frequencies with the largest magnitudes were selected. The number M is a function of the window size and will be discussed with the description of the quantization and coding scheme. Figure 9 shows a portion of the original amplitude spectrum of a window of voiced speech. Figure 10 shows which frequencies were chosen using the simple max-pick scheme.

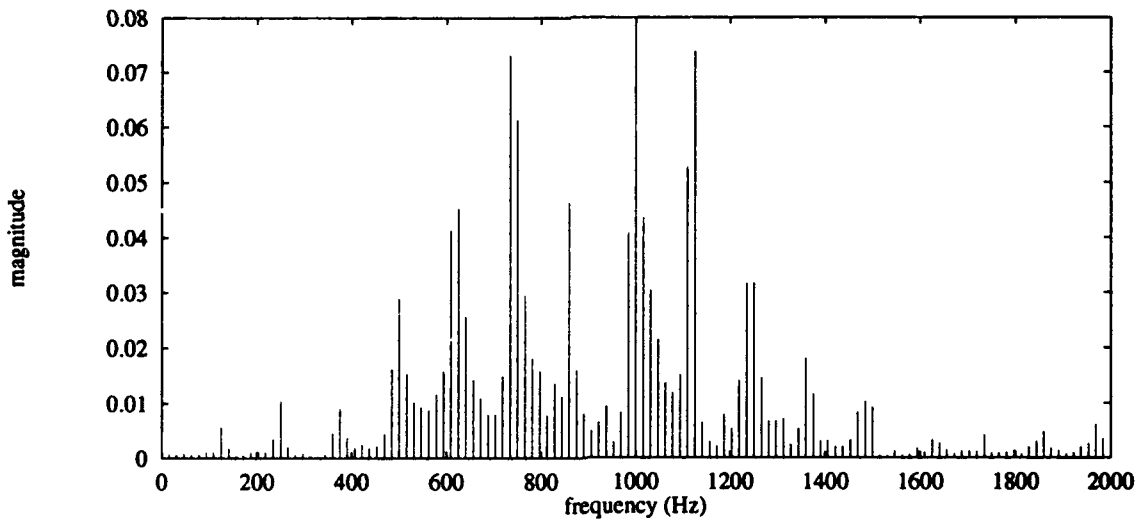


Figure 9. Original Voiced Spectrum.

The second scheme is again a max-picking scheme with the exception that frequencies above 600 Hz are boosted in order to give them a better chance of being selected. Frequencies are boosted according to the following equation:

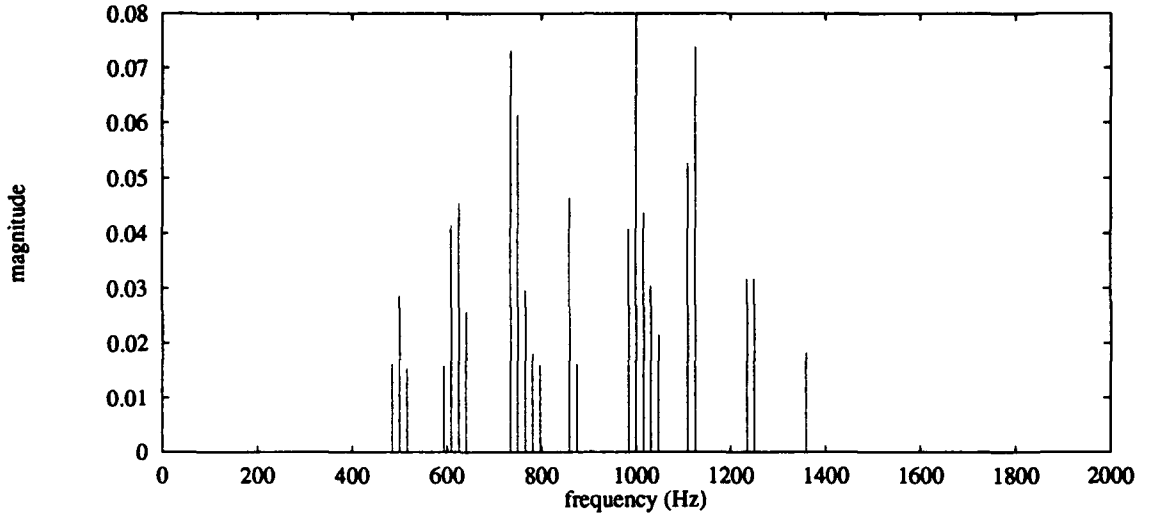


Figure 10. Maxpick Spectrum - Frequency components selected to represent the signal.

$$d[n] = \begin{cases} a[n], & 0 < n < N/2 \cdot F_b/F_s \\ a[n] \cdot n \cdot F_s/(F_b \cdot N/2), & N/2 \cdot F_b/F_s \leq n < N/2 \end{cases} \quad (2)$$

where $d[n]$ is the nth element of an $N/2$ element decision vector, $a[n]$ is the amplitude of the nth element of an N point DFT, F_s is the sampling frequency, and F_b is the frequency above which the amplitudes are boosted. For the system, F_s = 8 KHz, F_c = 600 Hz, and N = 512. Frequencies are selected by using a max-pick scheme on the elements of the decision vector $d[n]$. Figure 11 shows the frequencies that are chosen using the max-pick scheme with frequencies above 600 Hz receiving a 6 dB per octave boost in energy. Note in Figure 11 that more frequencies in the 1000 - 1400 Hz range are selected.

The third frequency selection scheme uses a peak-picking scheme. This peak-picking scheme is implemented by picking maximum values of the magnitude spectrum and eliminating the 12 nearest neighbors. The frequencies that are selected by this scheme are likely to be similar to those selected using the method chosen by Capt McMillan and Capt Ricart. In Capt McMillan's scheme (14), the glottal frequency is chosen and harmonics of the glottal are selected using a neighborhood rule. In Capt Ricart's coder, frequencies are

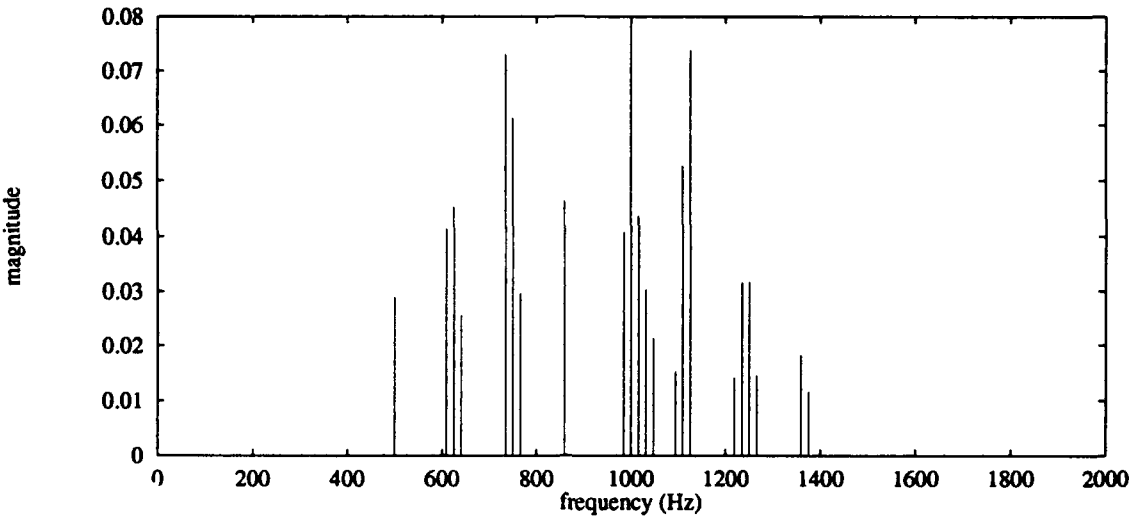


Figure 11. Maxpick with 6 dB boost Spectrum - Frequency components selected to represent the signal. Frequencies above 800 Hz are scaled to improve their selection chances.

selected using a lateral inhibition scheme in which neighbors within selected frequency bands compete to represent the band. Spectral peaks are selected in all three of these schemes. Figure 12 shows the frequencies that are chosen using the peak-picking scheme. This method samples the frequency domain more uniformly than the other schemes; however, since this method ignores many of the larger magnitude frequencies, it introduces more mean squared error than do the other schemes.

The fourth frequency selection scheme uses a max-picking algorithm also, except that in this case, selecting a frequency ωin eliminates it's four nearest neighbors from the selection process. This is done because when observing the amplitude spectrum of speech, frequencies neighboring peaks also have large amplitudes with respect to frequencies not in the neighborhood of peaks. Also, since more energy is contained in the lower frequencies, frequencies neighboring low frequency peaks tend to have larger amplitudes than higher frequency peaks. Because of this most high frequency information is ignored. Eliminating neighbors allows higher frequency peaks to be selected. Figure 13 shows the frequencies

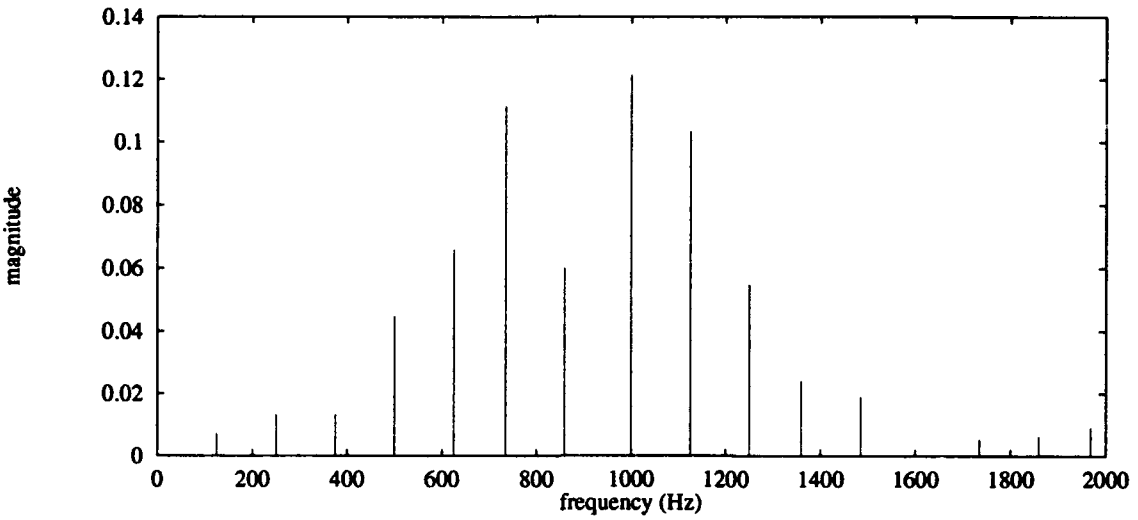


Figure 12. Peak-pick Selection Spectrum

that are chosen using the max-pick scheme when the four nearest neighbors of a selected frequency are eliminated from consideration.

A trade-off is involved when neighbors are removed from the selection process. The energy lost when neighbors are ignored contributes significantly to increasing the mean squared error between the original and the reconstruction. Not only does the reconstruction have a greater mean squared error, the sound quality of the reconstruction is also degraded.

In an attempt to recover some of the energy lost by disregarding the neighbors, an estimate of the amplitude of the neighbors is made. The shape of the spectrum near the peak is roughly approximated by a triangle. Since the peak amplitude of a selected frequency is transmitted, the spectrum can be filled in on the receiving end by setting the amplitudes of the two nearest neighbors to be equal to some constant times the amplitude of the peak. Since the two nearest neighbors of the peak values are eliminated on the analysis end, the next nearest frequency that could be selected is plus or minus three frequency bins away. If the nearest possible frequency bin is selected, then its nearest two neighbors are also set equal to it's amplitude times a constant. The resulting spectrum can look quite jagged.

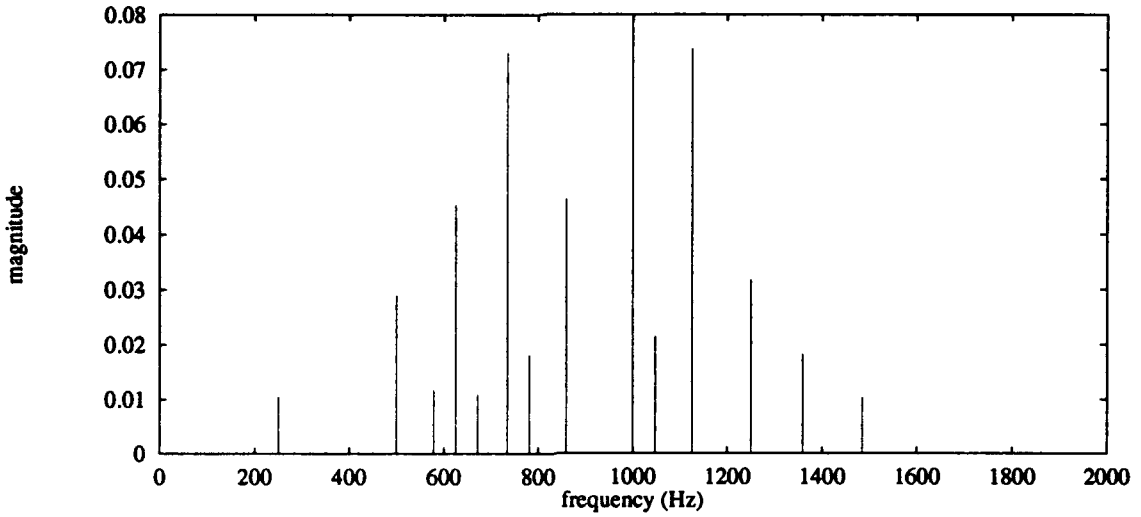


Figure 13. Maxpick with Neighborhood Exclusion Spectrum - Frequency components selected to represent the signal. The four nearest neighbors of a selected peak are eliminated from consideration.

In order to smooth out this jagged spectrum, an interpolation scheme is used to adjust the amplitudes of frequencies that are between a selected frequencies and estimated neighbor frequencies. Figure 14 shows a portion of the jagged spectrum before interpolation, while figure 15 shows what a smoothed version of the same spectrum looks like. Compare the original spectrum in Figure 16 to the jagged and interpolated spectra in Figures 14 and 15.

Estimating the magnitudes of the neighbors helps return some of the energy to the spectrum, but unless the phase information for the estimated neighbors is included, then the energy will be distributed quite differently in the time domain. Although the amplitude spectrum (Figure 16) shows that amplitudes in the neighborhood of peaks are related, the phase plot shows that the phase of neighbors does not appear to be related to the phase of the selected peaks. Figure 17 shows the phases of a portion of the spectrum of a voiced speech sound, the phases corresponding amplitude peaks are highlighted. In order to not mutilate the spectrum of the signal, the phases of the two nearest neighbors to a selected peak are transmitted along with the phase of the peak value itself. Figure 18 shows

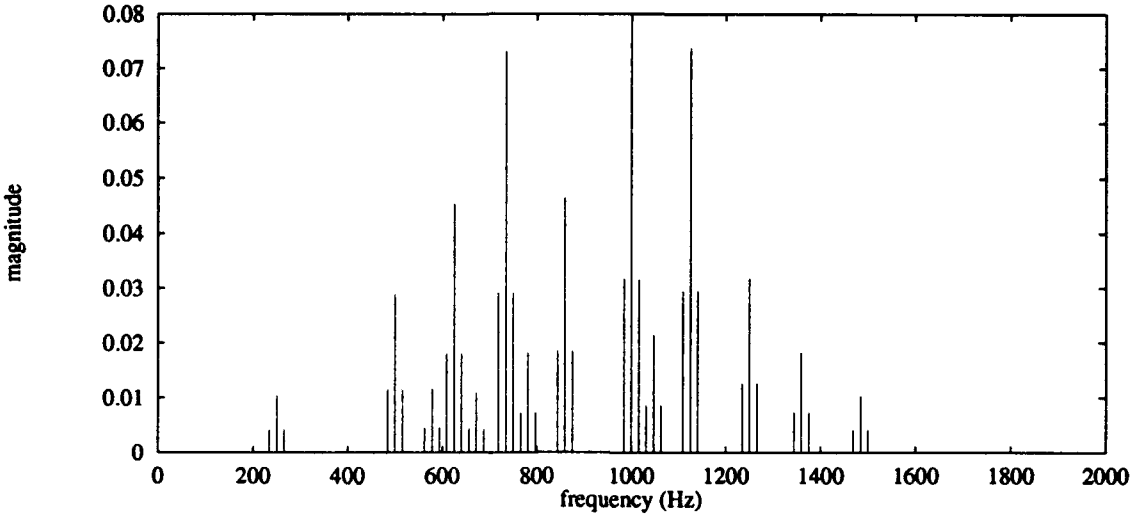


Figure 14. Maxpick with Neighborhood Exclusion Spectrum - Neighbors estimated. Frequency components selected to represent the signal. The four nearest neighbors of a selected peak are eliminated from consideration. Amplitudes of two nearest neighbors are estimated.

the transmitted phases of a portion of the spectrum of a voiced speech sound for which the amplitudes of the peaks' neighbors are estimated. A trade-off is made by sending this additional phase information, namely fewer frequencies can be sent for a particular window. This trade-off will be explained in the discussion of quantization and coding. Although fewer peak frequencies can be sent, by estimating the amplitudes of neighboring frequencies, and more complete, although partially estimated, frequency spectrum can be generated on the receiving end.

Figures 10, 11, 12 and 15 show which frequencies would be chosen to represent the original spectrum for each of the different selection schemes. For comparison, the original spectrum is shown in Figure 9. The frequencies selected by all four of these schemes do a fairly good job of representing speech which is not noiselike. If the speech is noiselike, a sparse spectrum does not do an adequate job of representing the signal. The next section describes how the system handles noiselike speech.

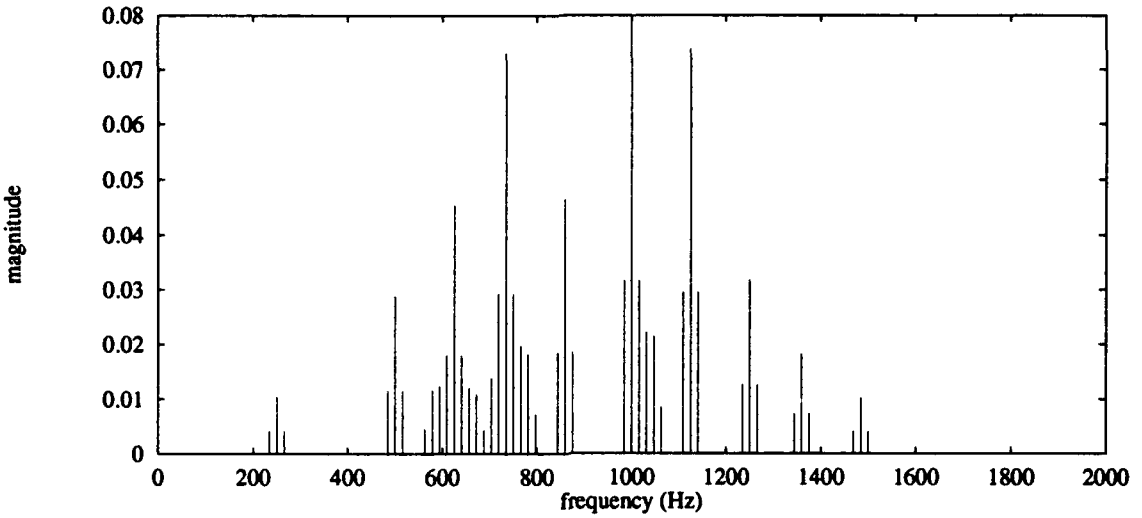


Figure 15. Maxpick with Neighborhood Exclusion Spectrum - Neighbors estimated with interpolation. Frequency components selected to represent the signal. The four nearest neighbors of a selected peak are eliminated from consideration. Amplitudes of two nearest neighbors are estimated, and interpolation is performed between the estimates.

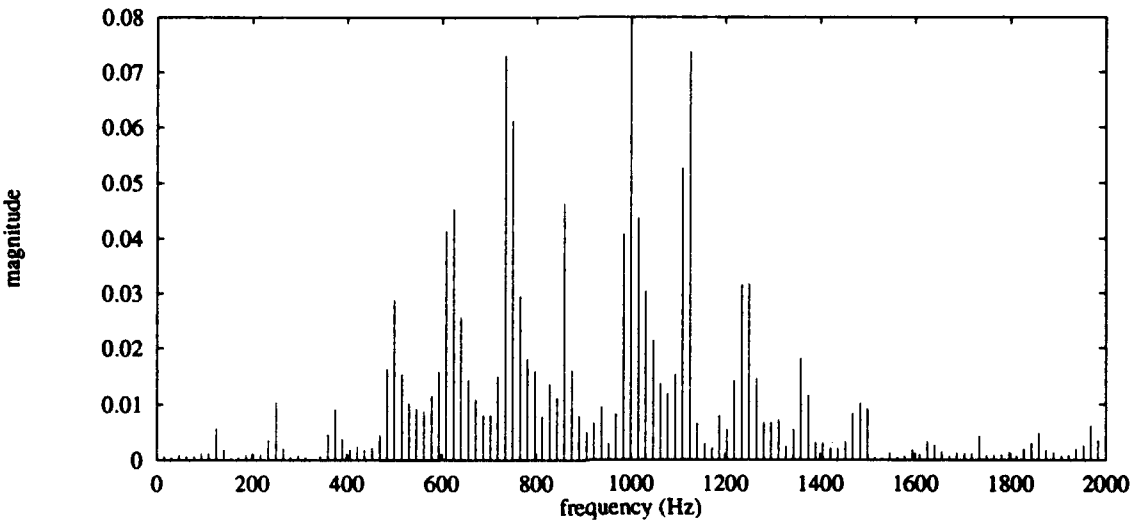


Figure 16. Original Voiced Spectrum - repeat of Figure 9.

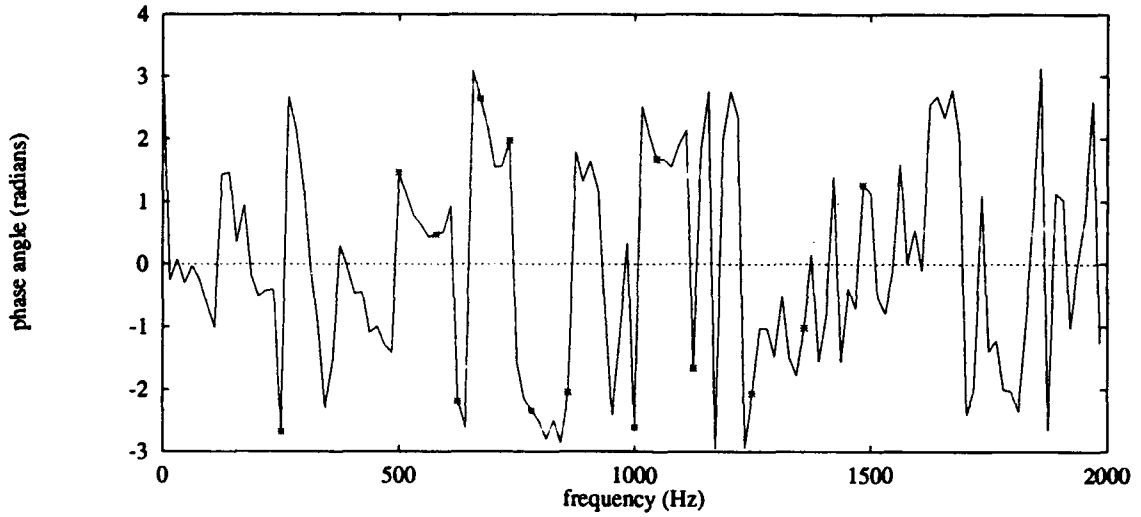


Figure 17. Phase Spectrum of a Voiced Time Slice - No apparent relationship between consecutive phase values. Phase components that match selected frequencies are highlighted.

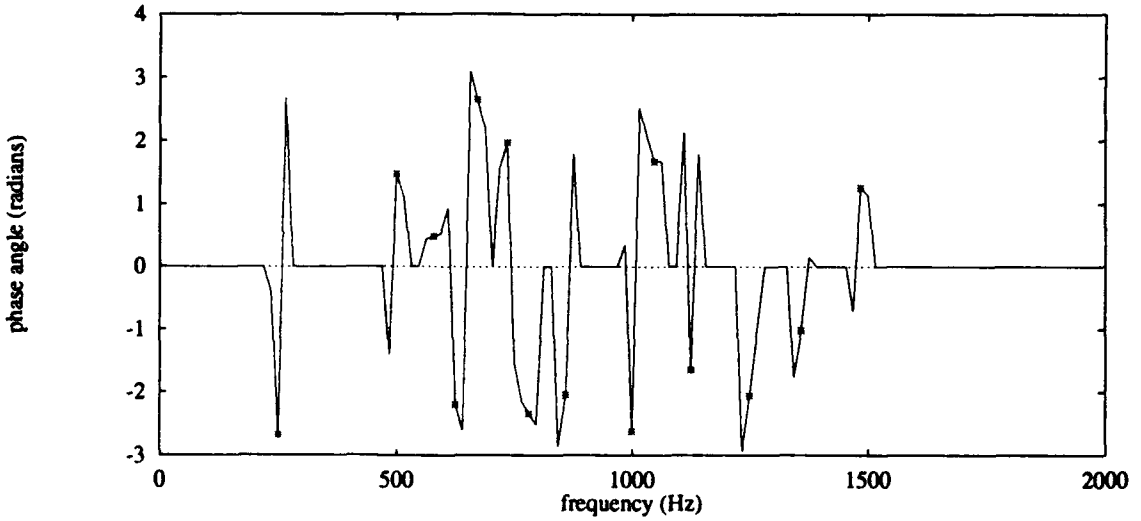


Figure 18. Transmitted Phase Spectrum of a Voiced Time Slice - Phase value of the amplitude peaks nearest neighbors are sent. Phase components that match selected frequencies are highlighted.

Handling Noiselike Speech Most speech coding methods handle noiselike speech differently than they handle periodic speech. In LPC systems, the filter is excited with either the glottal pitch or noise source. In the McAulay/Quatieri (20) system, a voicing probability is computed, and the phase of certain components is randomly set based on the likelihood that a given slice of speech contains a noiselike component. Noiselike speech is also handled differently than periodic speech by this speech coding system.

It was observed, that in order to recreate a good replica of noisy speech by sending sparse spectrum, at least 75 percent of the original frequencies must be sent. Since at a maximum the system can send about 10 percent of the frequency components, another method must be used to represent the noisy parts of speech. Several different methods were tried to fill in the missing components of the sparse spectrum that was trying to represent a noiselike sound. One method attempted to fill in the missing components by performing a linear interpolation between the selected frequency components, then scaling the interpolated components by a random value between 0 and 1. Another method tried filling the spectrum by convolving the selected frequencies with exponential functions, and again scaling the filled in values by multiplication with a random value between 0 and 1. In both cases, the phase components were randomized. None of the spectrum filling methods produced acceptable sounding fricatives. All of the reproduced sounds had an artifact that produced a kind of "metallic" sound. Since attempting to fill in the spectrum produced unacceptable results another method for representing the noiselike sounds was tried. In the new method, the energy in the noiselike signal was computed and transmitted for variably spaced frequency bands. On the receiving end, the energy per band was reproduced, and the resulting noiselike sounds no longer contained the "metallic" sounding noise.

The reason the energy is broken down into bands, rather than computed for the entire slice of speech is that different noiselike sounds have energy concentrated in certain frequency ranges. For example, the unvoiced fricative ʃ (as in "sh") has more energy in frequencies above 2 KHz, while the unvoiced fricative h (as in "he"), has most of it's energy in a band around 1000 Hz. Although the exact spectral components are not

necessary to create a reasonable replica for these sounds, the energy concentration within their spectra is.

As a first attempt, the energy was computed for 16 spectral bands. The bands were of variable widths based on an approximation of the mel scale. Fant (18:80) provides the following equation which approximates the mel scale:

$$mel(f) = \frac{1000}{log(2)} \cdot log(1 + \frac{f}{1000}) \tag{3}$$

where f is the frequency in Hz.

Table 1 lists the frequencies bands into which the spectrum was divided. Equally spaced mel scale bands lead to unequally spaced frequency bands.

Table 1. Mel Scaled Noise Frequency Bins

$mel(f)$	f	FFT bin numbers
0-145	0-106	0-7
146-290	107-220	8-14
291-435	221-350	15-22
436-580	351-500	23-32
581-725	500-650	33-42
726-870	651-825	43-53
871-1015	826-1020	54-65
1016-1160	1021-1235	66-79
1161-1305	1236-1470	80-94
1306-1450	1471-1735	95-111
1451-1595	1736-2020	112-129
1596-1740	2021-2340	130-149
1741-1885	2341-2680	150-172
1886-2030	2681-3090	173-197
2031-2175	3091-3520	198-225
2176-2320	3521-4000	226-255

On the receiving end, a file containing the frequency domain representation of white Gaussian noise is maintained. When energy information for a noiselike sound is received,

this file is accessed and a noisy spectrum is obtained. The spectral components are scaled so that the energy of each frequency band on the analysis portion of the system is preserved.

This scheme was used to reproduce all of the sounds that the neural net decision structure determined to be noiselike. These sounds included the fricatives, the stops, and background noise. For unvoiced fricatives and unvoiced stops this scheme worked quite well; however, for the noiselike sounds that also contained voiced information, the scheme did not accurately reproduce these sounds. The voicing information of the voiced fricatives and stops was completely lost. Voiced fricatives such as v (as in "vote") sounded just like their unvoiced counterparts, such as f (as in "for"). For voiced fricatives and voiced stops, the voicing information is vital for a good reproduction of the original sound. Since few bits are needed to send the banded energy information for noiselike sounds, the remaining bits can be used to send selected spectral components. The same frequency selection schemes described earlier can be used to pick frequencies for speech that is noiselike. Adding these components to the noisy spectrum generated using the banded energy information produced voiced fricatives and voiced stops that sounded much more like the original.

In order to be able to send more frequency components, the number of frequency bands was decreased from 16 to 8. Decreasing the resolution of the banded energy information had no perceptible affect on the sound of the reconstructed noiselike speech. It is possible that the frequency bands need not be mel scaled. Equally spaced frequency bands may do just as well; however, since the ear tends to pay attention to pitch according to the mel scale, and since nothing is gained from a computational standpoint by using equally spaced frequency bands, there was no obvious reason to try them.

At this point in the system, the speech has been partitioned into windows and characterized as either noiselike or periodic. Based on the determined type of speech present, the correct representation for the window has been obtained. The next step is to quantize the frequency and noise energy information. The next section covers the quantization and encoding process.

Quantization and Encoding Quantizing and encoding are complimentary activities. The encoding scheme determines how many bits are available for each of the parameters that will be transmitted. The available bits per parameter determine the quantization levels. The encoding scheme and the quantization together determine how many frequency components can be transmitted for each slice of speech. Since the analysis and synthesis windows vary in length from one time slice to the next, yet a constant data rate is desired, it is convenient to consider the information for each slice of speech in terms of a frame of data. Each frame will have some header information, and then depending upon the length of the analysis window and the type of speech being processed, it will contain a variable amount of frequency and noise energy information. Longer time slices of speech will provide more frequency information.

For the coding scheme used by this system the header of each frame must contain the noiselike/periodic decision, analysis window length, maximum amplitude of all frequency components sent, and the bin number, amplitude and phase of the lowest frequency transmitted. The reasons for including noiselike/periodic decision and the analysis window length in the header should be obvious. The rest of the header information requires explanation.

The maximum amplitude of all frequency components sent is needed so that all of the amplitudes can be quantized in comparison to the maximum amplitude sent. The amplitude values are determined from the magnitude of the FFT's real and imaginary parts. Since the DSP chip scales outputs between -1.0 and +1.0, the resulting amplitudes are between 0 and 1. Recalling that the entire input sequence had to be scaled before the FFT process, most of the amplitudes are close to zero. To keep the small amplitudes from all being quantized to the same level, it is necessary to quantize them relative to another small value. Quantizing the amplitudes in this manner decreases the quantization error.

One way to determine which frequency bins are being sent is to send the bin number. Since there are 256 possible bins, this would require 8 bits for each frequency component sent. This is obviously an expensive way to send the bin information. A more economic

method, is to send a number that is the difference between two consecutive bins. In order to start this process, the number of the lowest selected bin is sent. From this starting point all the other selected bin locations can be determined, and instead of sending 8 bits to identify each selected bin, the difference can be encoded in 5 bits. Only using 5 bits to determine the offset between bins means that selected bins can be no more than 32 units apart. If two consecutive bins happen to lie more than 32 bins apart, a place holder bin is set up at the 32nd bin. The amplitude and phase of this place holder bin are set to zero. The place holder bin will not contribute to the spectrum of the reconstructed signal. Frequency bins beyond the place holder bin will still contribute to the reconstruction. Based on a limited number of observations, selected bin spacing is usually closer than 32, and the place holder is not often used.

Table 2 describes how bits are allocated to the different parameters that constitute a frame.

Using the values provided in the table, the quantization levels are set, and the number of frequency components that can be sent per frame can be determined. Using the above listed values, phase can be quantized into 16 levels, amplitude can be quantized into 8 levels, and the noise energy band information can be quantized into 16 levels. The noise energy is quantized with respect to the energy contained in the frequency components sent. The energy of the transmitted components is computed and the ratio of the energy per noise frequency bands to the total energy is the value that is quantized. As with the amplitude quantization, the resolution of the noise energy information is increased. Using the frame description, one can determine how many frequency components can be sent for each window size. The header information requires either 28 or 36 bits per frame depending on the frequency selection scheme. The noise information requires 32 bits per frame. With a desired data rate of 4800 bits per second, each millisecond of speech is allowed 4.8 bits. Since the analysis windows grow in increments of 4 milliseconds, each extension of the window increases the number of bits that can be sent by 19.2. The number

Table 2. Transmission Frame Description

Header Information	
Parameter	Number of bits required
Noiselike/periodic decision	1
Analysis window length	4
Maximum amplitude of all frequency components sent	10
Smallest frequency bin number	6
Smallest frequency bin amplitude	3
Smallest frequency bin phase	4
**Smallest frequency bin - 1 phase*	4
**Smallest frequency bin + 1 phase*	4
Transmitted Frequency Information	
Parameter	Number of bits required
Offset from previous frequency bin	5
Amplitude	3
Phase	4
**Frequency bin - 1 phase*	4
**Frequency bin + 1 phase*	4
Noise Energy Information	
Parameter	Number of bits required
Energy per frequency band	4
**Required for neighborhood exclusion scheme only*	

of frequency components M that can be sent per frame is governed by equation 4.

$$M = 1 + \frac{19.2L - (H + E)}{F} \tag{4}$$

where L is the length of the analysis window, H is the number of bits required by the header, E is the number of bits required to quantize the noise energy, and F is the number of bits required to quantize the frequency bin, amplitude, and phase of each component. The constant 1 is due to the fact that the header contains the frequency information of the lowest frequency sent. For the window sizes used in this system, L, is an integer between 4 and 16. The header size, H, depends on the frequency selection scheme, and is 28 for max-pick, max-pick with high frequency boost, and peak-pick, and 36 for the max-pick with neighborhood exclusion. The bits required for noise energy quantization, E, are either 0 or 32, depending on the type of noisiness of the particular window. The amplitude, and frequency bin information requires either 12 or 20 bits, depending on the frequency selection scheme. More bits are required if the max-pick with neighborhood exclusion scheme is used. Tables 3 and 4 list the number of frequency components sent for each scheme.

Although apparently less information is being sent using the neighborhood exclusion rule, since the phase is being sent and the amplitude being estimated for the two nearest neighbors, the reconstruction spectrum actually has about 1.75 times as many frequency components as the max-pick and max-pick with boost schemes.

Any quantization scheme will introduce some error to the reconstruction of the signal. Trade-offs must be made between the allocation of bits between the different parameters. A trade-off exists between the number of frequency components that can be sent and the number of quantization levels for the amplitude and phase information. McMillan's system used 8 frequency components, 2 amplitude bits, and 3 phase bits for each 40 milliseconds of speech. Using a variable window technique, the system under discussion could get up to 24 frequency components, with 3 amplitude bits and 4 phase

Table 3. Number of Frequency Components Sent - Max-Pick and Max-Pick with 6 dB/octave Boost

Analysis window size	Number of frequency components (non-noisy window)	Number of frequency components (noisy window)
16	5	2
20	6	3
24	8	5
28	9	6
32	11	8
36	13	10
40	14	11
44	16	13
48	17	14
52	19	16
56	21	18
60	22	19
64	24	21

Table 4. Number of Frequency Components Sent - Max-Pick with Neighborhood Exclusion

Analysis window size	Number of frequency components (non-noisy window)	Number of frequency components (noisy window)
16	3	1
20	4	2
24	4	3
28	5	4
32	6	5
36	7	6
40	8	7
44	9	8
48	10	9
52	11	10
56	12	11
60	13	12
64	14	12

bits for 64 milliseconds of speech . Since McMillan's system used an overlap and add method for reconstruction, his effective number of frequency components may have been larger. Through testing, McMillan determined that the number of phase quantization levels seemed to affect the quality of the reproduced speech more than did the number of amplitude quantization levels. Informal listening test performed on the system under discussion agrees with his finding.

Decoding Decoding is accomplished by reversing the encoding process. Since the header provides the window size and periodic/noiselike decision parameters, the decoder knows how to extract the spectral information. The window size and periodic/noiselike decision determine the number of frequency components sent, thus the decoder knows how many frequency components to extract and whether or not the noise energy information is present. The header provides the bin number, amplitude, and phase of the lowest frequency sent. All other frequency bin numbers are determined by frequency bin difference information. The magnitude scaling information is also contained in the header. The amplitudes and phases of the transmitted frequencies are determined by converting the quantized levels back into floating point values. The decoder also fills in the spectrum for noiselike sounds. Based on the amount of energy in each frequency band, the decoder scales the spectrum of a white Gaussian noise file and adds it to the frequency information that has been received. The result of the decoding process is a 256 point complex vector that represents the reconstruction spectrum. The reconstruction spectrum will be transformed back to the time domain.

Transforming the Frequency Domain Signal Back to the Time Domain The reconstruction spectrum is transformed back to the time domain by using the hardware FFT of the DSP chip. The amplitude and phase complex spectrum is converted to a real and imaginary complex spectrum to accomplish the FFT process. The resulting time domain speech signal is truncated to match the analysis window size.

Concatenating and Smoothing the Truncated Speech Signals As was previously noted, by not using the overlap and add method for reconstruction, discontinuities can exist near the window borders. Simple concatenation of the resulting time slices of speech introduces a "popping" noise. Apparently this "popping" noise is due the discontinuities at the window borders. In order to reduce these discontinuities, a smoothing process is performed on the samples near the window borders. The smoothing process is described in the following paragraph.

Beginning eight samples prior to the window border, and extending eight samples beyond the window border the following smoothing function is employed.

$$\begin{aligned} x[n] &\leftarrow x[n] - (x[n] - x[n+1])/3 \\ x[n+1] &\leftarrow x[n+1] + (x[n] - x[n+1])/3 \end{aligned} \tag{5}$$

where $x[n]$ is the reconstructed speech sequence. Figure 19 shows the results of the smoothing process. The solid line shows the speech without smoothing, the dashed line shows the speech after smoothing. Smoothing attenuates the "popping" noise; however, it does not totally eliminate it.

Error Correction The reproduced speech signal is obviously not an exact replica of the original. Eliminating spectral components has an adverse affect on the quality of reproduced speech. The whole process of selecting frequency and quantizing components, and simulating noiselike speech is definitely a nonlinear process. The question is "Is there some way that errors introduced by this grossly nonlinear process can be corrected?" Since neural networks have shown ability to learn and predict functions, it is proposed that a network could be trained to correct the errors introduced by the system.

A couple of attempts were made to try and teach a recurrent neural network to correct the errors introduced by the system. The first attempt was a single input network. The input to the network was the time domain reconstruction of an entire utterance. Figure 20 shows the network architecture. Unlike the network used to make the periodic/noiselike

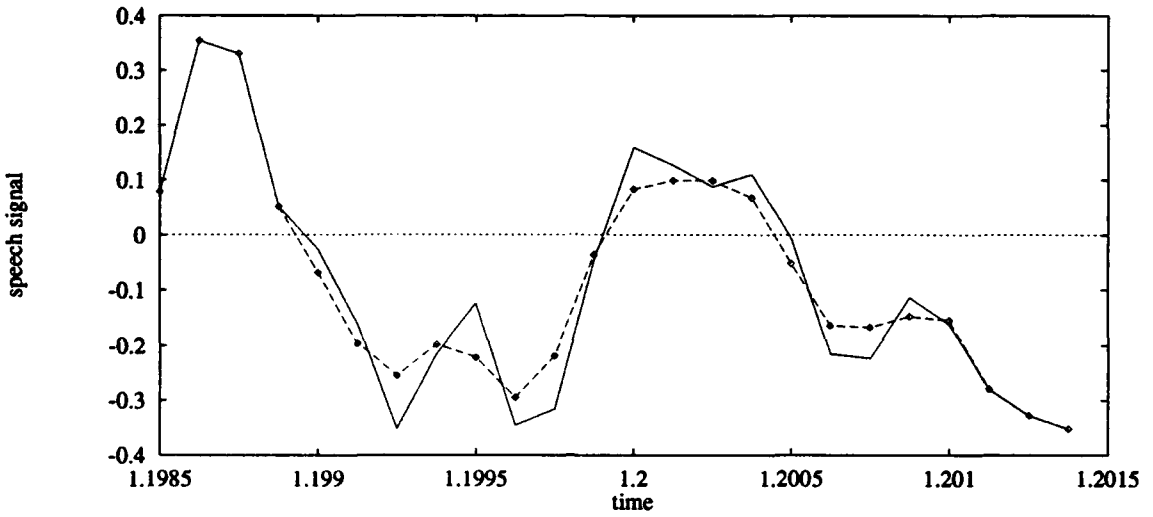


Figure 19. Smoothed Version of the Reconstructed Signal - the solid line is the initial reconstruction, the dashed line is the smoothed version.

decision which used a sigmoidal output, this network has a linear output. The network was trained by setting the desired output equal to the time domain signal that was originally input to the system. After the network was trained to a level at which the error between the input and output signal approached a constant value, the weights were fixed, and the network architecture was incorporated into the system. In informal listening tests, listeners reported that speech produced by passing the reconstructed signal through the neural net, sounded noticeably better than did the uncorrected reconstruction. The "popping" noises were further attenuated, and other artifacts of the process seemed to be corrected. The error corrected reconstruction did not; however, reach the level of quality of the original recorded speech.

The next attempt at correcting the errors introduced by the system was to include three time samples as inputs to the networks. The inputs were the current time sample, plus the two previous time samples. Figure 21 displays the architecture of the network. This error correction network did not attenuate the "pops" as did the single input network. Also other artifacts remained in the reproduced speech.

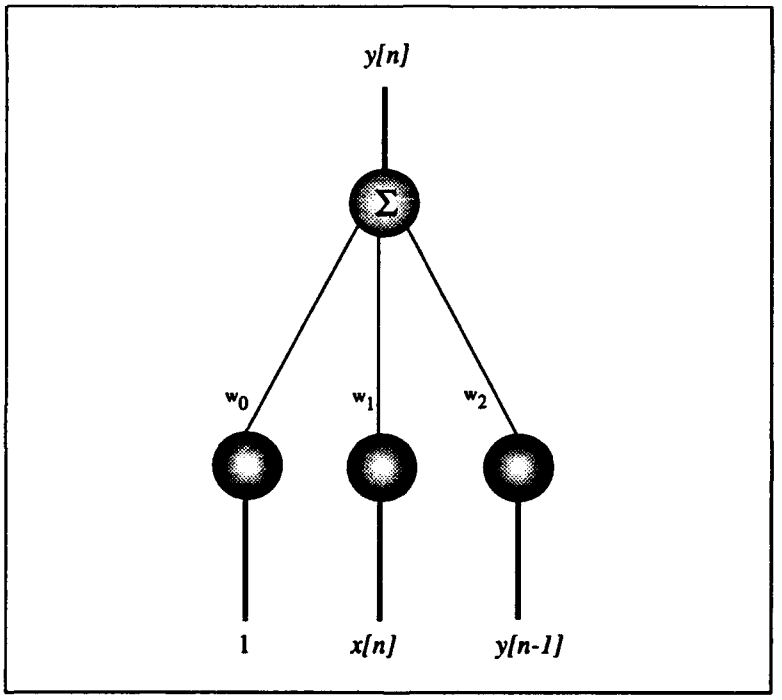


Figure 20. Error Correction Neural Network - One Input. This network tries to correct the time domain reconstruction of the signal. The network is trained by comparing reconstructed signal to the original. Notice that the output is linear, not sigmoidal.

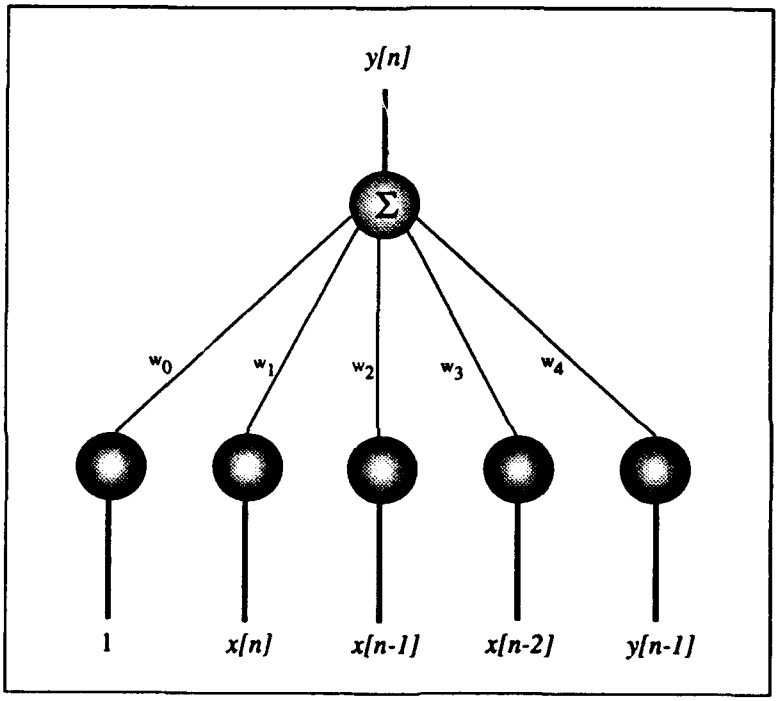


Figure 21. Error Correction Neural Network - Three Input. The inputs to the network are the current sample and it's previous two values. The network is trained by comparing reconstructed signal to the original.

Looking at the networks from a digital signal processing point of view, they appear to be time domain filters. The networks weights correspond to digital filter coefficients. If a time domain signal is fed into the network and the output is computed by taking a weighted sum of the input, bias, and previous output then an IIR filter is achieved. The weights of the filter are obtained by forcing the output of the network to look like the original signal. Figure 22 shows the neural network when analyzed as a time domain filter. Due to the bias term used in the recurrent network, analyzing it as a time domain filter introduces nonlinearities. To simplify the analysis, a new signal $\tilde{x}[n] = x[n] + 1 \cdot w_0/w_1$ is introduced. The new signal $\tilde{x}[n]$ is the original signal $x[n]$ plus a small DC bias. The DC bias term w_0/w_1 is on the order of 10^{-3}, and can be ignored. Table 5 lists the weights obtained from training the recurrent network on speec produced using the max-pick selection scheme.

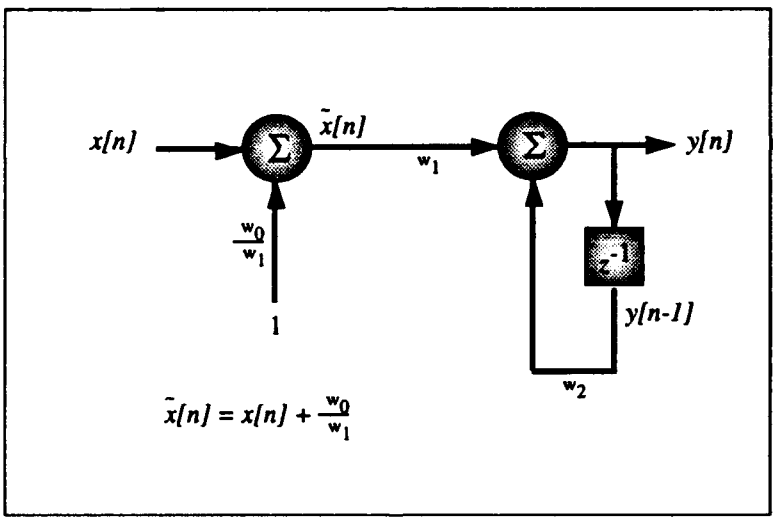


Figure 22. Error Correction Neural Network as an IIR Filter - One Input.

The frequency response of the network was computed. It turns out that for the max-picking frequency selection schemes, the network acts like a high-pass filter. Since the max-picking frequency selection schemes all tend to pick lower frequencies, it makes intuitive sense that the network would try to correct errors by boosting the higher frequency content of the signal. For the peak-picking scheme, the frequency response of the filter

Table 5. Error Correction Network Weights - One Input

Weight	Value
w_0	-0.001630
w_1	1.173535
w_2	-0.283277

had the shape of a very flat high pass filter. This also makes intuitive sense because the freqency domain was sampled much more uniformly in the peak-picking scheme. Figure 23 shows the frequency response of the neural network for the max-pick selection scheme.

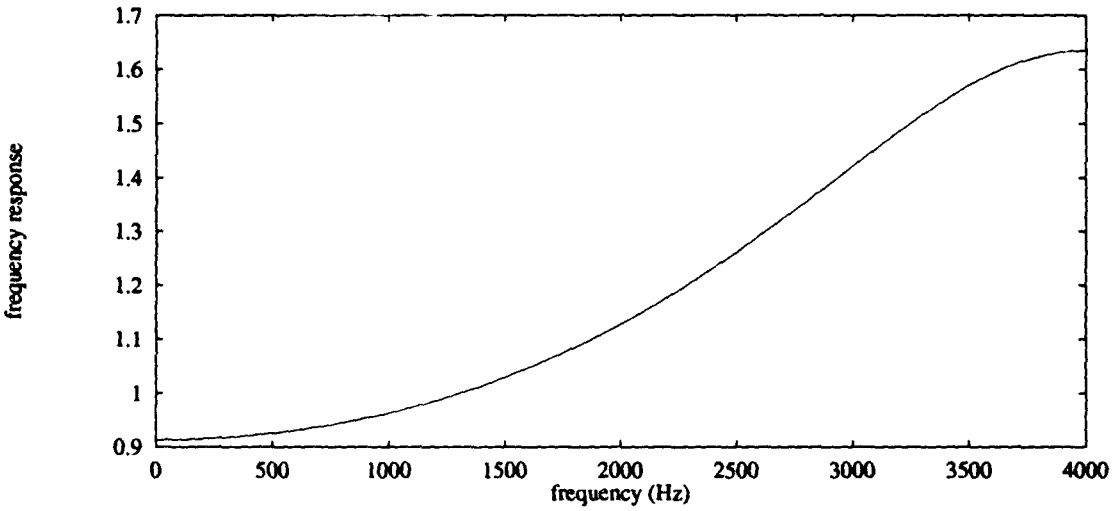


Figure 23. One Input Error Correction Neural Network - Frequency Response.

Using a single input to the network allows for a very simple filter. To increase the complexity of the filter, delayed versions of the input signal were also fed into a three input network. Figure 24 shows the network architecture as a time domain filter. The network weights are listed in Table 6. The frequency response of the filter is plotted in Figure 25. As suspected, the frequency response of the three input filter is more sophisticated than that of the one input filter. Surprisingly, the three input filter did not produce better sounding speech than did the single input filter.

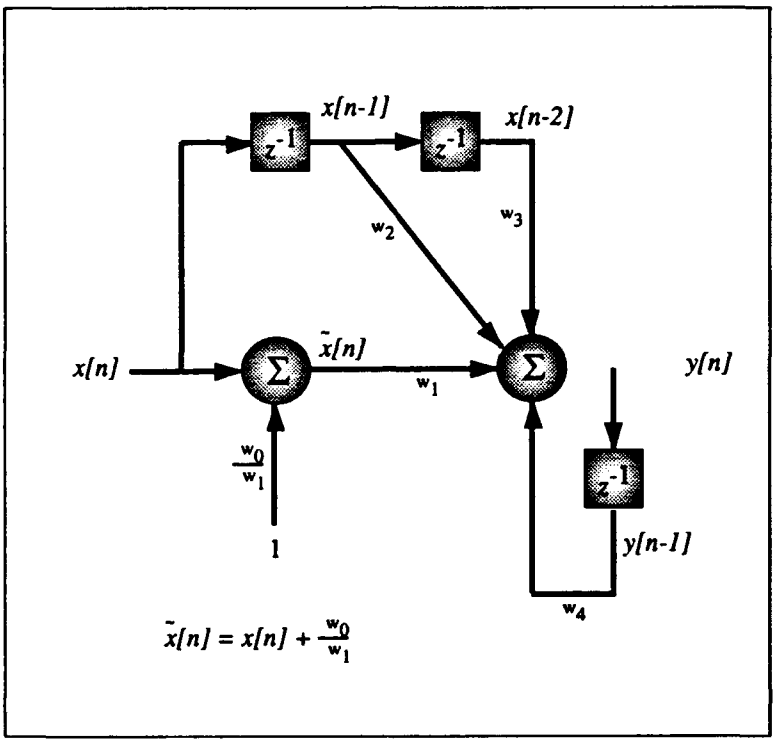


Figure 24. Error Correction Neural Network as an IIR Filter - Three Input.

Table 6. Error Correction Network Weights - Three Input

Weight	Value
w_0	-0.004104
w_1	0.959943
w_2	-0.006950
w_3	0.673271
w_4	-0.638417

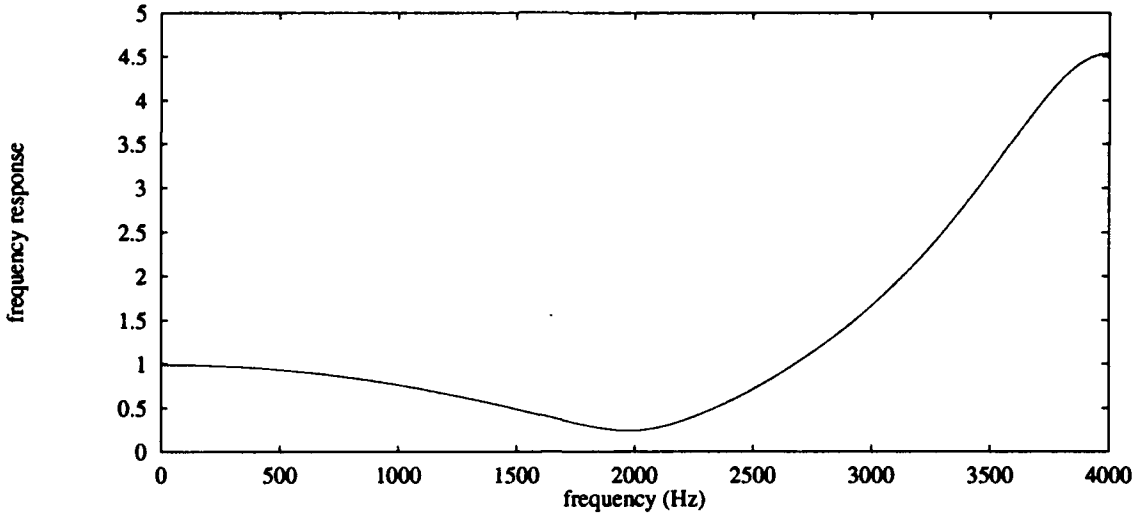


Figure 25. Three Input Error Correction Neural Network - Frequency Response.

Since this filter is obtained by training the weights, the training process could be considered a filter design technique. Using few delay elements for both input and output greatly limits the complexity of the kind of filter which can be obtained. If more time delayed inputs and outputs are added to the system, then more sophisticated filters should be possible. Besides using all linear nodes, one could include some nonlinear nodes in the network also. Perhaps the nonlinear nodes could be trained to better model the nonlinear processing which the system does to a signal. Due to limited time, these questions are beyond the scope of this thesis.

Two other attempts at correcting the errors produced by the system were made. The first is an attempt to perform an error correction on a each time slice of speech processed by the system. The network architecture used to perform this correction is the same as that of Figure 20. Instead of training the network on the entire signal and fixing the weights, this error correction method would train the network on a time slice of speech that corresponds to an analysis window. Each frame of speech to be transmitted would then also include the weights for the network. Training the network on the entire signal amounts to performing a global correction, training the net on smaller portions of the signal amounts to performing

local corrections. This scheme has definite drawbacks from an implementation point of view. Training the network takes time, and for each weight sent the amount of frequency information that could be put in a frame decreases. In practice, for the weights to be transmitted, they would have to be quantized. It is not known how sensitive the network is to slight changes in the weights.

A stepwise approach to implementing this error correction scheme in the system was taken. If the local corrections would buy an increase in sound quality, then training, quantization, and network weight encoding would be considered. The number of training epochs was set to 200. This means the net would see the entire input signal 200 times. For the global correction scheme 200 epochs was enough for the network to approach a constant error value, so that value was chosen for training the net for the local correction scheme. Training the network for 200 epochs for individual time slices of speech and using the network weights to make corrections on these time slices did not improve the quality of the reproduced speech. Since the first step of this process was unsuccessful, no attempt was made to quantize the weights or to analyze the network's sensitivity to quantization.

Since most of the nonlinear processing takes place in the frequency domain, it was proposed that the frequency domain may be the best place to perform the error correction. The problem with attempting to perform the error correction in the frequency domain is in the presentation of the data to the network. Recurrent networks have shown a capacity to learn temporal sequences when a relationship exists between samples in the sequence. No such relationship seems to exist between consecutive frequency domain samples of speech.

The only attempt to correct the speech in the frequency domain was to train the network on a sequence of amplitude components. The amplitude spectrum used to reconstruct the periodic speech were concatenated and fed into the network. The actual amplitude spectrum of the periodic speech were concatenated and used as the desired output of the network. The network was unable to learn any relationship between the two. No error correction was attempted on the phase spectra.

Another possible frequency domain error correction scheme was suggested. In this scheme, there would be 512 parallel recurrent networks. Each of the recurrent networks would be responsible for tracking one of the frequency bins corresponding to the output of a 512 point FFT. Half of the networks would attempt to correct the amplitude components, and the other half would correct the phase components. The inputs to the networks would be the amplitude and phase information of the selected frequency components from one time slice to the next. The desired outputs would be the actuai amplitude and phase information. Such a system may be able to track errors introduced in the frequency domain from one time slice to the next. Again, due to time constraints, this error correction scheme was not attempted.

A question of generalization versus memorization often arises when using neural network technology. For example, did the network learn the corruption done by the system, or did it just learn how to correct the errors introduced by the specific speaker. The network was trained on a small set of utterances by one speaker. The resulting network seemed to improve the quality of reproduced speech for other utterances by the same speaker, but the network was not tested on other speakers.

The work on error correction using recurrent networks raises more questions than it answers. Since it seemed to improve the quality of the reproduced speech, without affecting the bit rate, it was included in the system design. The final stage of the system is the postprocessing of the signal.

Postprocessing Postprocessing of the signal is a two step process. First, the signal is converted from the floating point format used throughout processing to a 16 bit linear format. The NeXT computer plays back the 16 bit linear speech by converting it from digital to analog on the CODEC. The resulting output can be heard from the NeXT built-in speaker or over a set of headphones. The NeXT computer also has a dual output jacks. These jacks will be used to feed into a reel-to-reel tape recorder to produce a testing tape.

Summary

This chapter covered the design of the speech encoding system. It covered the process from recording the speech to playback of the reconstructed signal. It detailed the periodic/noiselike decision process, the frequency selection process, the quantization scheme, the window boundary smoothing process, and the error correction scheme. Justifications for design decisions are provided throughout the chapter. The following chapter explains the testing process for speech signal generated by the system using various frequency selection schemes. The tests will also judge the success of the recurrent network error correction scheme.

IV. System Testing

Introduction

When testing speech coding systems, one usually is interested in two measures – quality and intelligibility. Although these measures may be related, they are not identical. Obviously, if the quality is high, then so should also be the intelligibility. The converse is not true, intelligible speech is not necessarily high quality speech. For instance, most speech synthesis systems for personal computers produce speech that is quite intelligible, but has a definite synthetic sound which degrades the quality. The goal for coding systems is to produce speech that is not only intelligible, but also possesses a high degree of quality. Quality measurements are subjective in nature, while intelligibility measures are more objective. Speech quality is an attribute that is hard to quantify. The processing of the auditory system is not well understood. Speech that sounds good to one listener may sound terrible to another. Not only that, the quality of a given system may seem to improve after listening to speech produced by a specific coder. The auditory system tends to adapt to the type of distortion introduced by a coder. Intelligibility can, on the other hand, be given a quantifiable measure. Intelligibility is often measured using tests in which rhyming words are the presented to listeners. The listeners indicate which of the possible rhyming words was perceived. The number of words perceived correctly provide a score that indicates the intelligibility of a system.

Testing employed for this system was a two stage process. The first stage of testing attempted to provide a quality rating, and the second stage of testing provided an intelligibility rating. Quality testing was done to determine which system design should be tested for intelligibility. Quality testing was performed by volunteer AFIT listeners. These listeners provided a quality score for systems that employ different frequency selection schemes with and without the error correction recurrent network. The system which obtained the highest score was used to generate a test set for intelligibility testing.

Intelligibility testing was performed by the Biocommunications and Bioacoustic Branch of the Armstrong Laboratory at Wright-Patterson AFB.

Quality Testing

The purpose of quality testing is twofold. Quality testing provides a subjective measure of ability of the system to reproduce speech that sounds natural to the listener. Sampling and digitizing speech introduces distortion to the original analog signal. Windowing the digitized speech and selecting a sparse set of components from the frequency domain further degrades the quality. Quantizing the sparse frequency domain representation of speech does even more damage to the quality of the speech signal. Different kinds of distortions that are introduced affect speech perception differently. Clipping speech introduces one type of distortion, while low pass filtering speech introduces another. The goal of any system is to keep the perceptible distortion to a minimum. The first purpose of quality testing is to rate the reproduced speech against some standard. The second purpose of quality testing was to decide which system to use to produce intelligibility test data base.

In order to measure the quality of the reproduced speech, volunteers were asked to listen to several different utterances. The utterances came from the following categories: original speech, digitized speech, or encoded speech. The original speech is a recorded version of a tape provided by the Biocommunications and Bioacoustic Branch. The digitized speech is the recorded version of the same speech sampled at 8 KHz on the NeXT computer system. The encoded speech consists of recordings of speech generated using the four different frequency selection schemes with and without the recurrent network error correction processing. The encoded speech is recorded directly from the NeXT. The total number of different versions of the speech is ten. The original version and digitized version are provided to render a baseline by which the encoded versions can be judged.

Procedure A test tape was generated that contained 12 utterances from each of the following categories:

1. original speech
2. digitized speech
3. system 1 - max-pick speech
4. system 2 - max-pick speech with recurrent network error correction
5. system 3 - max-pick with 6 dB/octave boost
6. system 4 - max-pick with 6 dB/octave boost speech and recurrent network error correction
7. system 5 - peak-pick speech
8. system 6 - peak-pick speech with recurrent network error correction
9. system 7 - max-pick with neighborhood exclusion speech
10. system 8 - max-pick with neighborhood exclusion speech and recurrent network error correction

Volunteers were asked to rate the speech on a scale of 1–10. A rating of 1 would indicate the listener could not understand the speech, while a rating of 10 would indicate that the speech sounded as though it were coming from someone speaking to them in a quiet room. The volunteers were given the following instructions:

> Thank you for participating in this test. The test will last approximately 15 minutes. The goal of this test is to compare several samples of human speech that have been created using a speech encoding system. Speech coding introduces various artificial sounds. The goal of this coding system is to reproduce speech in an efficient manner while limiting distortions. The tape contains several utterances generated by speech coders which use different representations of speech. Some of the utterances are the original recordings

from which the reproductions were made. Your job is to rate the quality of the utterances.

You will hear 120 separate utterances with a space of approximately 3 seconds between each. Please rate each utterance on a scale of 1 to 10. A rating of 1 should indicate that the speech is totally unintelligible. A rating of 10 should indicate that the speech sounds as if it were coming from a person speaking directly to you in a quiet environment. Please place your ratings on the score sheet provided. When you are ready to begin, press the play button on the recorder. You will hear a brief message. While the message is playing you can adjust the volume of the recorder. When you have finished the test, please return the score sheet to Captain Switzer.

Table 7 is a sample of the score sheet used for the test.

The mean score per system is used to determine which of the systems provide speech with the highest rated quality. The scores for each of the systems, as well as the original and sampled speech are provided in chapter V. The next section discusses the intelligibility testing.

Intelligibility Testing

The Biocommunications and Bioacoustic Branch of the Armstrong Laboratory at Wright-Patterson AFB performed the intelligibility testing. The standard test used by the branch to test intelligibility is the Modified Rhyme Test (MRT). The MRT is performed by presenting a series of utterances to a group of trained listeners who are employed by the branch. The utterance consist of a carrier phase and the rhyme test word. The MRT vocabulary consists of groups of words that sound similar. The words may have different initial consonants or different final consonants. After hearing to the phrase,"You will mark *blank*, please",where *blank* is the test word, trained listeners are to choose one word from a list of six possible words. Table 8 is a partial set of words used for a typical MRT. A score is provided for the system by giving the percentage of correct responses according to the following equation:

Table 7. Speech Quality Test Score Sheet

Utterance Number	Score (1-10)	Utterance Number	Score (1-10)	Utterance Number	Score (1-10)	Utterance Number	Score (1-10)
1		31		61		91	
2		32		62		92	
3		33		63		93	
4		34		64		94	
5		35		65		95	
6		36		66		96	
7		37		67		97	
8		38		68		98	
9		39		69		99	
10		40		70		100	
11		41		71		101	
12		42		72		102	
13		43		73		103	
14		44		74		104	
15		45		75		105	
16		46		76		106	
17		47		77		107	
18		48		78		108	
19		49		79		109	
20		50		80		110	
21		51		81		111	
22		52		82		112	
23		53		83		113	
24		54		84		114	
25		55		85		115	
26		56		86		116	
27		57		87		117	
28		58		88		118	
29		59		89		119	
30		60		90		120	

$$P = \frac{C - W}{T} \cdot 100 \tag{6}$$

where P is the MRT score, R is the number of correct responses, W is the number of wrong answers, and T is the total number of utterances tested.

Table 8. Modified Rhyme Test Sample - adapted from (17:192)

	WORD LIST					
	A	B	C	D	E	F
Part 1						
1	BAT	BAD	BACK	BASS	BAN	BATH
2	TAB	TAN	TAM	TANG	TACK	TAP
3	PUN	PUFF	PUP	PUCK	PUS	PUB
4	TEACH	TEAR	TEASE	TEAL	TEAM	TEAK
5	HEAR	HEATH	HEAL	HEAVE	HEAT	HEAP
6	CUT	CUB	CUFF	CUP	CUD	CUSS
7	FILL	FIG	FIN	FIZZ	FIB	FIT
8	BUN	BUS	BUT	BUFF	BUCK	BUG
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
Part 2						
26	LED	SHED	RED	BED	FED	WED
27	RAW	PAW	LAW	JAW	THAW	SAW
28	SIP	RIP	TIP	DIP	HIP	LIP
29	WAY	MAY	SAY	GAY	DAY	PAY
30	WILL	HILL	KILL	TILL	FILL	BILL
31	BOOK	TOOK	SHOOK	COOK	HOOK	LOOK
32	PEEL	REEL	FEEL	HEEL	KEEL	ELL
33	DIG	WIG	BIG	RIG	PIG	FIG
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.

Drawbacks

There are a number of drawbacks to the testing used to judge this system. The first drawback is that the quality testing is performed by untrained listeners. The volunteers used for quality testing are AFIT students and faculty. Their ability to determine types of distortion often observed in speech coding systems may be limited, and thus their scores may be skewed.

The second drawback is that the the tapes used for both tests came from the same speaker. Due to limited resources of time, disk space, and cpu time, it was not feasible to generate speech tapes for multiple speakers. Accurate Modified Rhyme Testing generally requires at least three male and three female speakers. When using the results of the tests, it must be considered that testing a single speaker does not provide a completely general indication of the ability of the system.

Another drawback to the testing performed is that the speech that was used to test the system was not corrupted with background noise. The original tape from the Biocommunications Branch was recorded in an anechoic chamber. Any actual system would have to be robust in the presence of background noise. Again, due the the previously mentioned resource constraints, no formal attempt to determine the performance of the system for noise corrupted speech was undertaken.

Although these tests do have drawbacks, they do provide a useful way to measure speech created by the system.

Summary

This chapter provides the basic strategy employed for testing the quality and intelligibility of the system. The quality tests are subjective, and served to identify which frequency selection scheme produced the best sounding results. Quality testing also served to provide a measurement for the usefulness of the recurrent network error correction scheme. Finally, quality testing does give some indication as to the acceptability

of speech reproduced using this coding scheme. Intelligibility testing provides a solid measure of system performance. Since other speech coding methods have reported MRT and DRT scores, the MRT score is a measure by which this system can be judged against others.

V. Test Results

Introduction

The previous chapter discussed the type of testing performed on speech generated by the system. Both quality and intelligibility are measured. This chapter will provide the results of the testing.

Quality Testing

Fifteen subjects participated in quality testing that was described in the previous chapter. This section details the results of the quality test. Again, the purpose of this portion of testing was to determine which of the frequency selection schemes produced the highest quality reproduction and to give a quality score for the system. Additionally, the test will determine whether or not the error correction neural network improved the quality of the reproduced speech.

The means and standard deviations for each of the following categories are displayed in Figure 26.

1. original speech
2. digitized speech
3. system 1 - max-pick speech
4. system 2 - max-pick speech with recurrent network error correction
5. system 3 - max-pick with 6 dB/octave boost
6. system 4 - max-pick with 6 dB/octave boost speech and recurrent network error correction
7. system 5 - peak-pick speech
8. system 6 - peak-pick speech with recurrent network error correction
9. system 7 - max-pick with neighborhood exclusion speech
10. system 8 - max-pick with neighborhood exclusion speech and recurrent network error correction

Score histograms for the tested systems are provided in Figures 27-36. The speech produced by the system that used the max-pick with neighborhood exclusion rule, nearest neighbor estimation, with recurrent network error correction was rated as the highest (5.650). The lowest score (4.094) came from the peak-picking system, with recurrent network error correction. Not including the original and sampled speech, the mean score for the systems is 5.087.

The mean score for the original speech is 9.683, while the mean score for the sampled speech is 9.272. These values indicate that 8-bit μ-law sampling at 8 KHz does very little damage to the quality of speech. These scores also provide a standard by which the other system scores can be judged. The listeners consistently rated the original speech near 10.

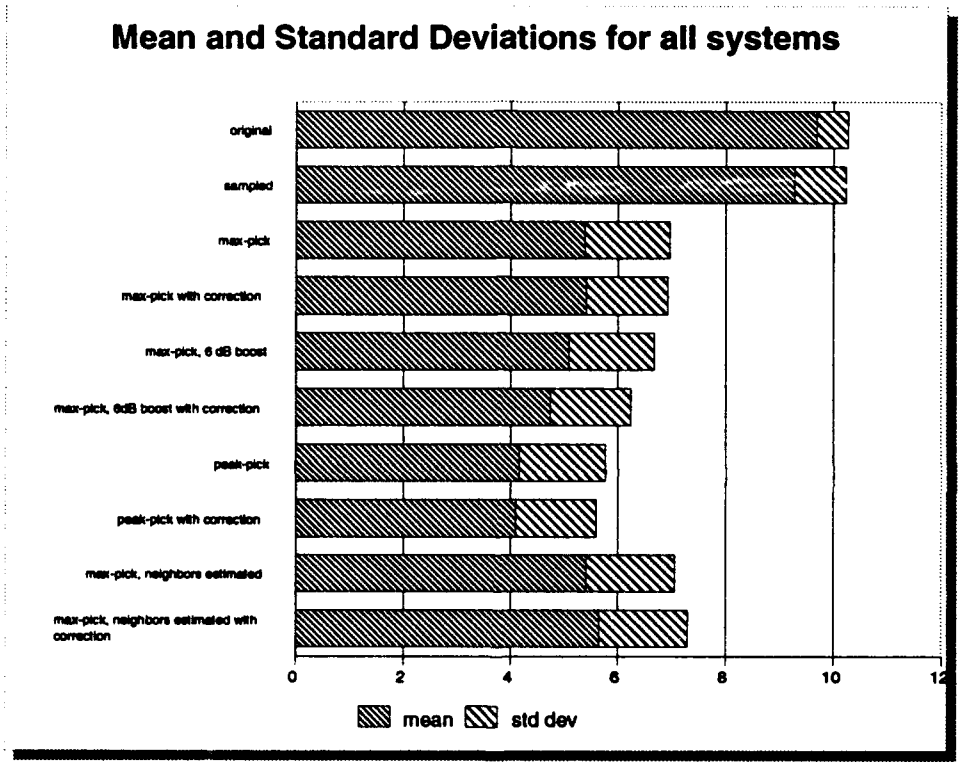


Figure 26. System Means and Standard Deviations - Note that the highest score (not including the original and 8 KHz sampled speech scores) is obtained by the system that pick maximums, estimates the values of the neighbors, and uses the error correction network.

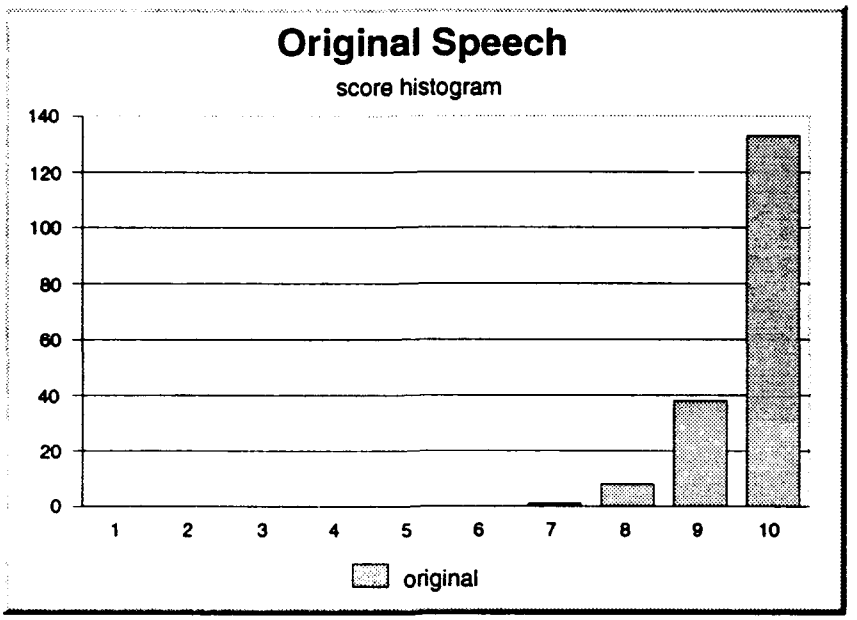


Figure 27. Original Speech Histogram - mean = 9.683, standard deviation = 0.582.

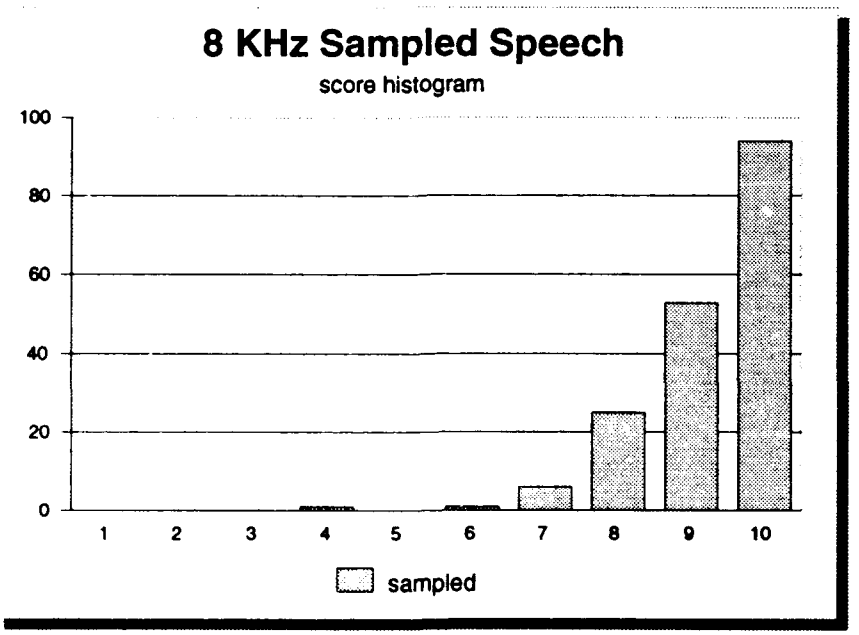


Figure 28. 8 KHz Sampled Speech Histogram - mean = 9.272, standard deviation = 0.953.

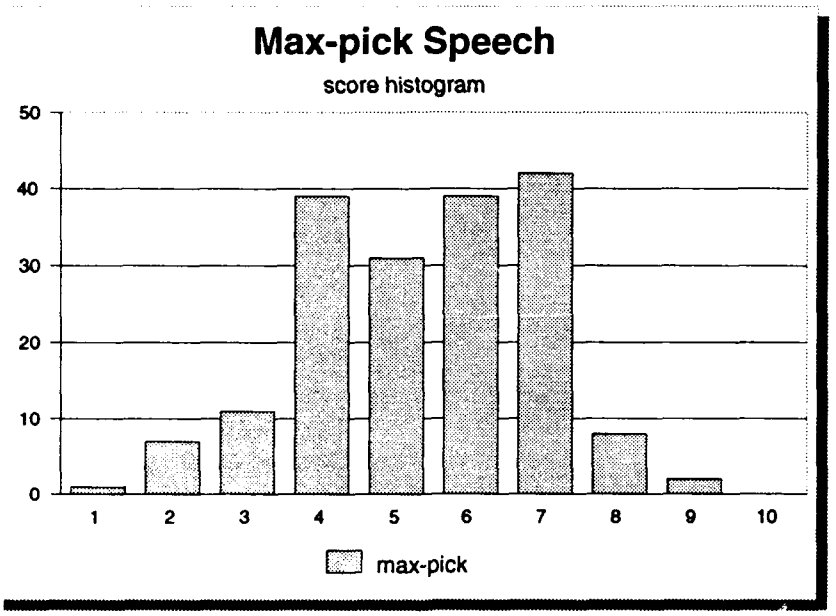


Figure 29. Max-Pick Speech Histogram - mean = 5.383, standard deviation = 1.575.

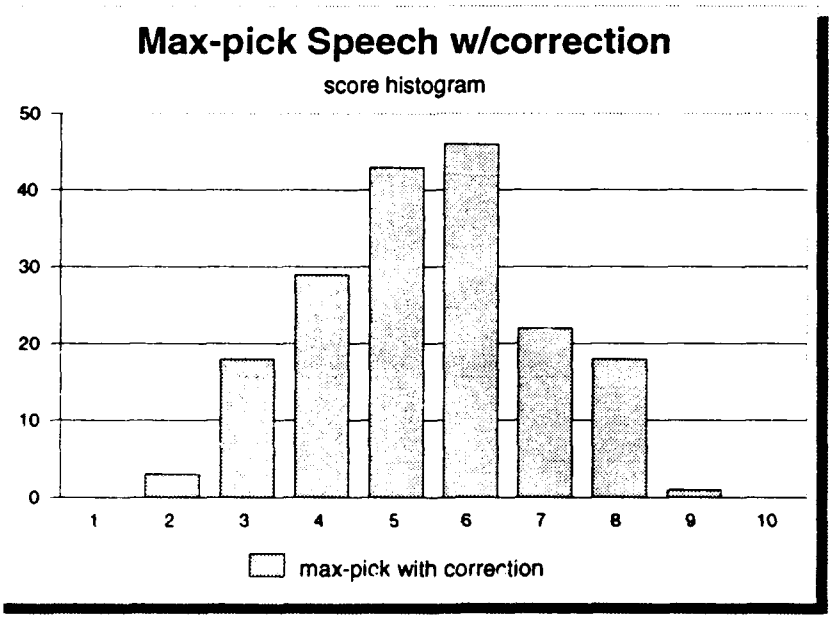


Figure 30. Max-Pick Speech with Error Correction Histogram - mean = 5.411, standard deviation = 1.508.

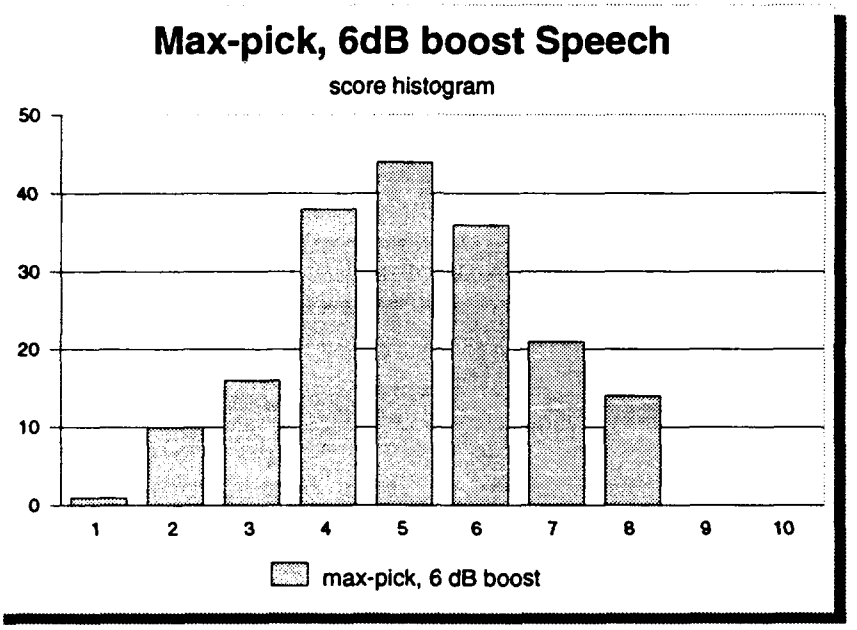


Figure 31. Max-Pick, Neighbors Excluded Speech Histogram - mean = 5.088, standard deviation = 1.586.

Determining the Best System The following question arises when deciding which system performed the best:

> Is the system with the highest mean significantly better than the other systems in a statistical sense?

In order to answer the question, one can employ a standard test using the t statistic. Such a test is described in Mendenhall, Wackerly, and Schaeffer (15:454-459). Observing the histograms for each system and applying the Central Limit Theorem, it is assumed that the distribution of scores approximates a Guassian. The null hypothesis H_0 then becomes:

> H_0: the means of Gaussian distributions for two systems are equal.

In order to reject the hypothesis with 99.5% confidence, the t statistic must be greater than 2.576. Table 9 presents the t statistics for all systems. Bold faced type in the table

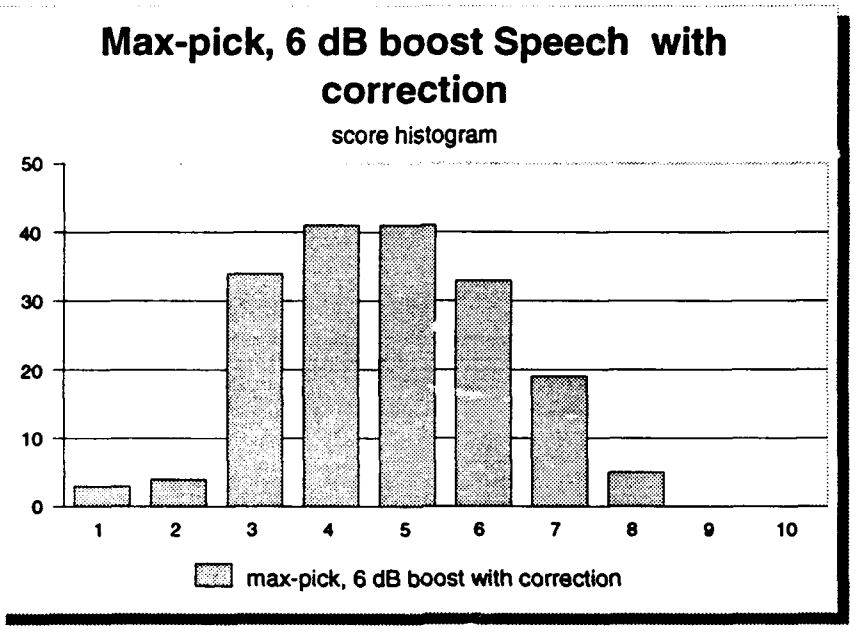


Figure 32. Max-Pick, Neighbors Excluded Speech with Error Correction Histogram - mean = 4.739, standard deviation = 1.496.

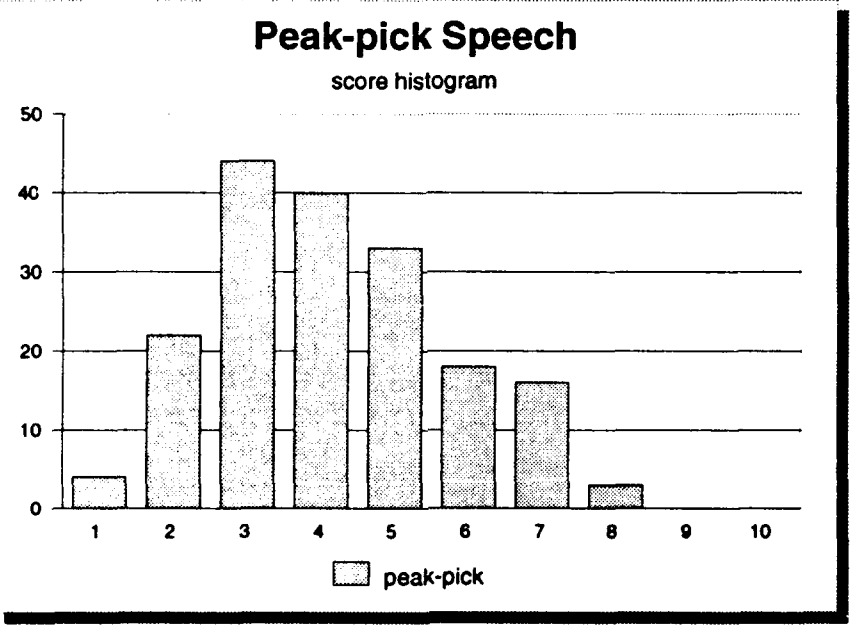


Figure 33. Peak-Pick Speech Histogram - mean = 4.161, standard deviation = 1.496.

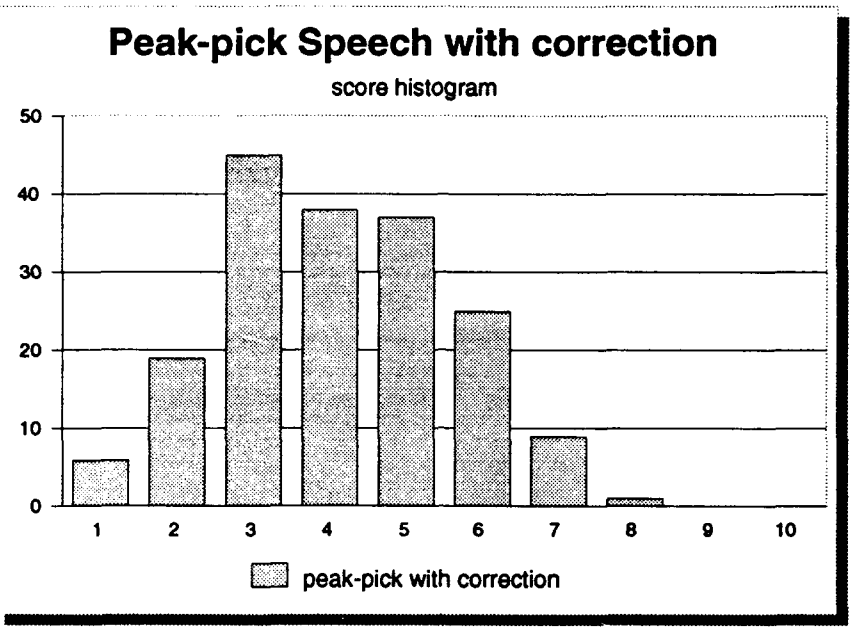


Figure 34. Peak-Pick Speech with Error Correction Histogram - mean = 4.094, standard deviation = 1.504.

indicates that the system means are statistically different. From Table 9, one can see that the highest scoring system (system 8) is only significantly better than systems 3,4,5, and 6. With a t statistic of greater than 1.282, we can reject the null hypothesis H_0, with 90% confidence. Using the 90% confidence level, system 8 is significantly better than all other systems. As such, system 8 was used to produce the MRT test set.

Error Correction Analysis An interesting result of the quality testing is found in the analysis of the systems that used the error correction recurrent network versus the systems that did not. Figure 37 summarizes the results. The mean score for the error correction speech systems was 5.100, as opposed to an mean score of 5.073 for systems without error correction. Comparing t scores of systems 1 with 2, 3 with 4, 5 with 6, and 7 with 8 in Table 9 will allow us to determine if the means of systems with and without error correction are significantly different. We can see we must fail to reject the null hypothesis for all cases using a 99.5% confidence level. This result is in conflict with the previous

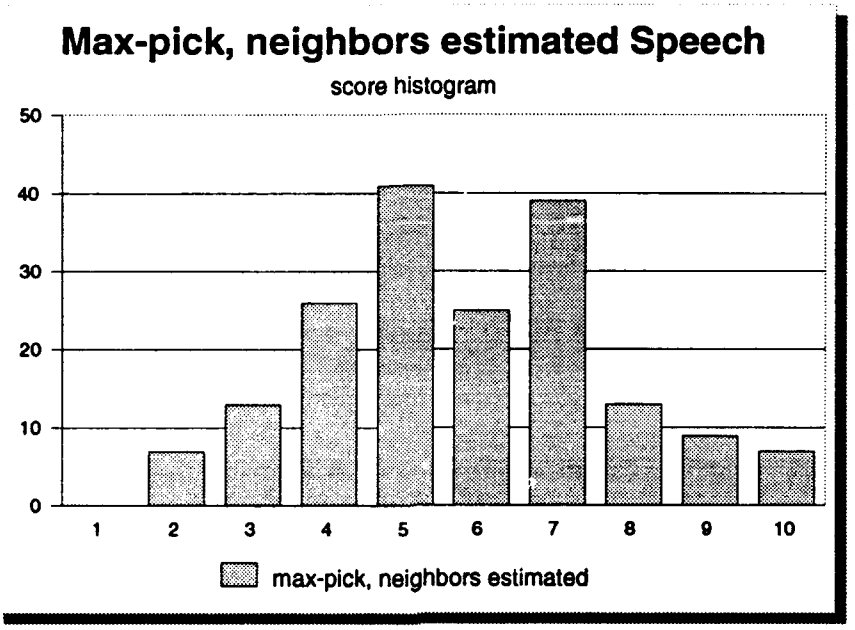


Figure 35. Max-pick, Neighbors Estimated Speech Histogram - mean = 5.411, standard deviation = 1.649.

Table 9. T-Test Scores - **Bold values** indicate statistically different means between the systems with a 99.5% confidence.

	system 1	system 2	system 3	system 4	system 5	system 6	system 7	system 8
system 1	0.000	0.171	1.768	**3.980**	**7.305**	**7.939**	0.163	1.569
system 2	0.171	0.000	1.975	**4.245**	**7.629**	**8.292**	0.000	1.435
system 3	1.768	1.975	0.000	2.154	**5.527**	**6.104**	1.890	**3.292**
system 4	**3.980**	**4.245**	2.154	0.000	**3.540**	**4.076**	**4.051**	**5.492**
system 5	**7.305**	**7.629**	**5.527**	**3.540**	0.000	0.407	**7.300**	**8.698**
system 6	**7.939**	**8.292**	**6.104**	**4.076**	0.407	0.000	**7.913**	**9.352**
system 7	0.163	0.000	1.890	**4.051**	**7.300**	**7.913**	0.000	1.375
system 8	1.569	1.435	**3.292**	**5.492**	**8.698**	**9.352**	1.375	0.000

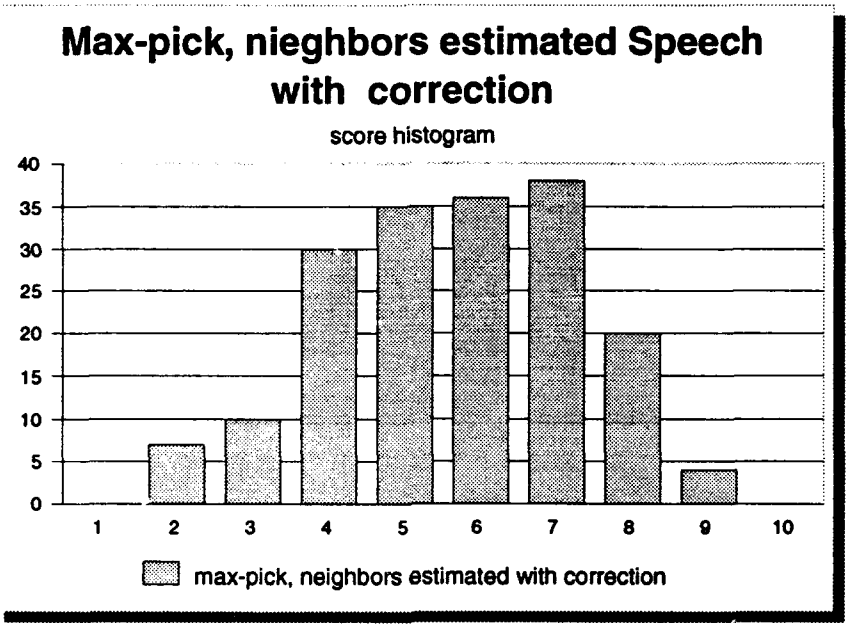


Figure 36. Max-pick, Neighbors Estimated Speech with Error Correction Histogram - mean = 5.650, standard deviation = 1.648.

informal listening test results. The error correction network does not seem to significantly improve the quality as previously believed.

There are a couple of possible explanations for these results. First, the network was trained on only one utterance. Perhaps the network learned only how to correct the errors introduced in coding one specific utterance. The second and more likely excuse is that during informal testing in the design of the system, somehow the listeners were biased to believe that the error correction network improved the speech. It may have been a case of what Dr. Kabrisky calls the "Clever Hans" syndrome. The "Clever Hans" syndrome goes something like this:

> A man has a horse named Hans who can solve simple math problems. On-lookers provide questions for the man to ask Hans. The man asks Hans the question, and before long Hans begins to paw the ground with his hoof the correct number of times to answer the question. One of the onlookers bets

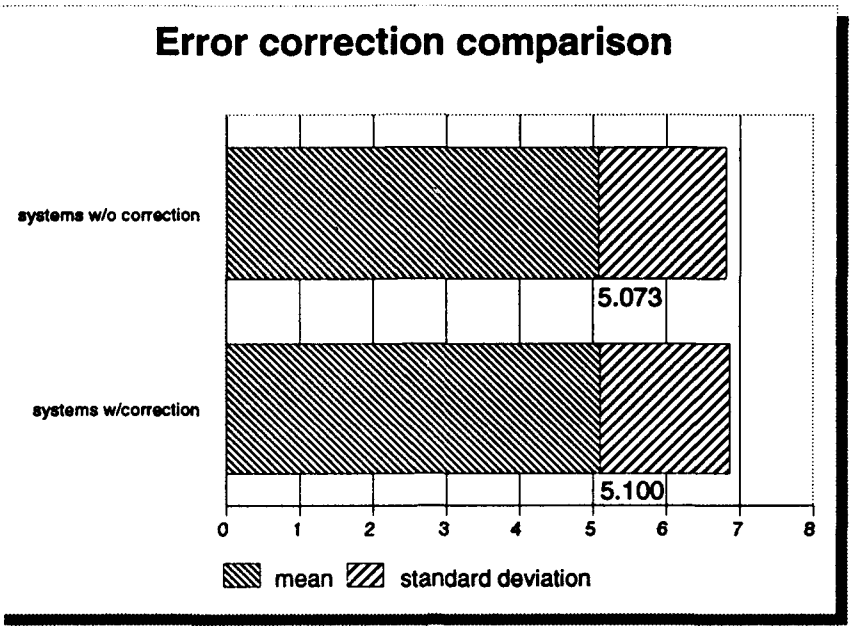


Figure 37. Error Correction Comparison - Scores for systems using the error correction network are lower.

> Hans' owner that Hans can't get the answer right if Hans isn't looking at his owner. Hans' owner accepts the bet and stands behind a sheet to ask Hans the question. Upon hearing his owner ask the question, Hans begins to paw the ground. Hans keeps on pawing right past the correct answer to the question. Hans' owner didn't realize it, but he had been cueing Hans when to stop all along. Not only did Hans' owner have to pay off the bet, he was forced to realize that his horse wasn't quite as smart as he thought.

In the informal listening tests, listeners were asked to rate two utterances. They were not told the difference between the two, only to rate which of the two sounded better. The listeners nearly always chose the error corrected version. Evidently, an unintentional, nonverbal clue was being given to the listeners.

Listener Adaptation It is also interesting to look at the mean score for each utterance. Figure 38 shows the resulting means for each utterance. The utterances were randomized in order to remove any sequencing pattern from the presentation of the test set. The mean

score for the first six utterances is 4.688, while the mean score for the last six utterances is 5.297. The utterances were not completely randomized; however. The lower number utterances were presented earlier in the test set than were the higher number utterances. Figure 38 shows that utterances presented later in testing tended to score better than those presented earlier in the test. There are likely two reasons for this result. First, listeners probably were unsure what kind of rating to give the utterances early in the test. After listening for a while, listener would gauge the quality of an utterance based on previous experience. The listeners, not being accustomed to judging the quality of reproduced speech, probably expected the reproductions to sound better than they did. As such, they gave low scores to the early utterances and after realizing that the coded speech quality would not get much better, they decided to raise their quality scale. Several of the listener reported changing their scale after listening for some time. One participant even marked on the score sheet the place where he adjusted his rating scale.

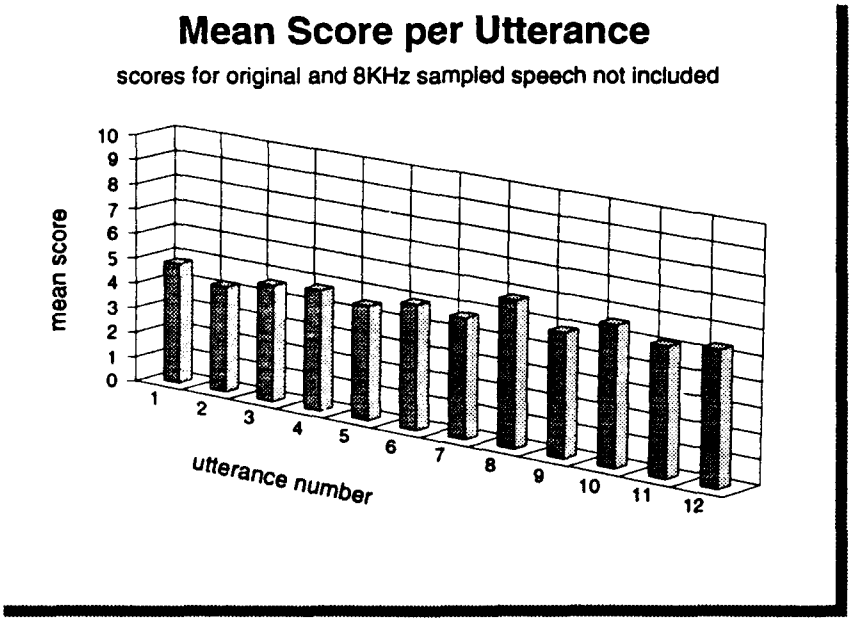


Figure 38. Mean Score per Utterance - Scores for utterances presented later in the test set tend to be higher.

Another explanation can account for the mean scores going up as the test went on.

The listeners probably adapted to the kinds of distortion that the were introduced by the coding scheme. Perhaps the auditory system began to provide some feedback that would block the distortion. The phenomenon of blocking out background noise is quite common in the auditory system. In everyday life, the ability to ignore background noise and focus on particular sounds is very important. Speaking over the telephone is a good example of this phenomenon. Telephone speech limited to frequencies between 300 and 3,300 Hz, with a great deal of channel noise added, along with environmental noise. The signal-to-noise ratio is low, yet we have no trouble at all blocking out the noise and focusing attention on the speech. Even with the glottal frequency filtered out of most telephone speech, we can nearly always understand what's being said as well as identify the speaker.

Intelligibility Testing

Intelligibility testing is still being conducted at the time of publication of this document.

Summary

This chapter discussed the results of system testing. The quality test results determined which of the system should be used for intelligibility testing. System 8 (max-pick with neighbor estimation and error correction) was determined to be the best. Other interesting results were uncovered, namely the systems that used error correction were not significantly better than those that did not, and that even for a relatively short test (about 20 minutes), listeners may have adapted to the distortion introduced by the system. Intelligibility testing was not completed before this document was published.

VI. Conclusions and Recommendations

Introduction

The previous chapter discussed the results of testing performed on speech generated by the system. This chapter will highlight some of the key points, and recommend possible enhancements.

Successes

This thesis was successful in several areas. To begin with, a highly intelligible 4800 bit per second coder was obtained. The coder incorporated some new ideas including the following:

- Transitions were detected using a recurrent neural network.
- Noisy sounds are hard to reproduce from a sparse spectrum; however, the noise energy information can be used for good sounding reconstruction of noisy speech.
- Several different frequency selection schemes were tested.
- Variable length analysis and synthesis windows were employed along with a simple window boundary smoothing scheme.
- Frequency information was quantized in a manner that allowed variable quantization levels depending on the magnitude of the largest frequency component sent.
- Recurrent neural networks were introduced as error correction devices.
- Simple recurrent networks were analyzed as time domain filters.

Each of these items can be considered as significant improvements on the previous frequency domain speech coders developed at AFIT.

Problems

There are also some problems that the system was not able to overcome. In the first place, a 4800 bit per second, "toll quality" speech coder was not obtained. Finding the transitions, did not solve all of the noise problems associated with McMillan's coder. In fact, the musical sound introduced by McMillan's coder seemed to be the same kind of sound found when to few frequency components were used to represent noisy speech. Each of the frequency selection schemes used in this thesis introduced some sort of distortion. It is possible that some of the added noise is a result of using rectangular windows. The smoothing scheme removes some of the noise; however, there may be other noise due to windowing that does not occur only at the window boundaries. Another problem associated with the system is that a noticeable increase in distortion is present when the selected frequency components are quantized. All systems sounded better when complete floating point information is maintained for the amplitude and phase of the selected components.

Recommendations

There are several possible changes that could be made to the system that may enhance its performance. Daubechies provides a method for constructed variably overlapping windows for the windowed Fourier transform (3:971-972). Using slightly overlapping windows may be the key to reducing noise that may be introduced by rectangular windows. Another possible enhancement could be increasing the number of quantization levels allowed for both magnitude and phase components. The tradeoff may mean fewer frequency components sent per frame; but, better quality speech may be obtained.

More complicated recurrent networks could be tried in order to reduce the errors introduced by this nonlinear system. I think more complicated networks including more feedback and nonlinear nodes may be able to improve sound quality. Besides using recurrent networks to improve this system, more research should be directed at studying them as digital filters, including recurrent networks with nonlinear nodes.

Conclusion

Further research could push the quality of the speech produced by this system beyond that of LPC and other sinusoidal coders.

Appendix A. *Speech Coding System Source Code*

This appendix contains a listing of the source code that implements the speech coding system.

```
/* Speech Coder                    */
/* Author: Capt Shane Switzer      */
/* Date:   31 July 1991            */
/* This program simulates a system for encoding speech.  The system takes a
   NeXT sound speech file sampled at approximately 8 KHz and stored in an 8 bit mu-
law format, and converts it to a 16 bit linear format for compression.  The 16 bit linear
data is converted to float and scaled to a range of [-1,1] for processing by the DSP chip.
The scaled file of floats is then transformed into the frequency domain via the DSP FFT
routine.  Frequencies are selected to represent the signal, and nonselected frequencies are
set to zero.  An energy conservation routine is used to boost the signal, and an inverse
Fourier transform brings the signal back into the time domain.  The user can choose to
window the input signal and to corrupt the input signal with noise.  If created, the noisy
signal is written to a file.  The reconstructed signal is converted from float back to 16 bit
linear, then back to 8 bit mu-law.  The resulting file is saved and is playable and viewable
using the NeXT SoundEditor program.

   To use the program:
   1. You must have a NeXT sound file.
   2. ENTER coder <infilename> <outfilename>, where infilename is the
      name of the input sound file, and outfilename is file to
      write the reconstruction to.
   3. When the program completes. The new sound can be viewed and played.  */

#import <sound/sound.h>
#import <math.h>
#import <dsp/arrayproc.h>
#import <mach.h>
#import <stdlib.h>
#import <stdio.h>
#import <objc/objc.h>
#import <soundkit/Sound.h>
#import <soundkit/soundkit.h>
#import <string.h>
#import <macros.h>

#define DATA_ADR DSPAPGetLowestAddressXY()   /* DSP FFT constants */
#define REAL_DATA DSPMapPMemX(DATA_ADR)
#define IMAG_DATA DSPMapPMemY(DATA_ADR)
#define LOOKUP_ADR DATA_ADR + COUNT
#define SIN_TABLE DSPMapPMemY(LOOKUP_ADR)
```

```
#define COS_TABLE DSPMapPMemX(LOOKUP_ADR)


#define COUNT 512            /* FFT length */
#define SLICE_SIZE 512       /* max time domain slice size */


#define PI 3.141592654
#define TWO_PI PI*2

#define ABS(x) sqrt(pow(x,2.0))
#define ABS2(x) ((float)sqrt(((double)x)*((double)x)))
#define SQUARE(x) ((x)*(x))

id mySound;
int slice_size;
long SOUND_SIZE;             /* size of the signal to be processed */
SNDSoundStruct *soundStruct,   /* NeXT Sound Structures for manipulating */
        *convertStruct, /* sound files */
       *newStruct;

typedef struct

 { int offset;

   int magnitude;

   int phase;

 }frequency_component;


typedef struct

 { int length;

   int noisy;

   int largest_magnitude;

   int base_frequency;

   int base_magnitude;

   int base_phase;
```

```
  int energy[8];

  frequency_component frequency_array[22];

 }frame_struct;



/******************** get_sound() ************************************
 This function opens and reads the signal values of a sound file.
 It converts the sounds to 16 bit linear and chops the sound at
 number of samples specified by SOUND_SIZE.  SOUND_SIZE is the length of the
 utterance modulo 32.
 CALLED BY: main
 CALLS: none
 INPUTS: temp - a pointer to an array of short ints, this is where
           the 16 bit linear version of the signal is kept
       infile - a string that specifies the input sound file name
 OUTPUTS: temp - a pointer to an array of short ints
*****************************************************************/

void get_sound(short **temp,char *infile)
{
  int error;
  error = [mySound readSoundfile:infile];
/* initialize mySound to infile's mySound object */
  soundStruct =[mySound soundStruct];
  [mySound isEditable];
  SOUND_SIZE = soundStruct->dataSize - soundStruct->dataSize % 32;
  SNDAlloc(&convertStruct,SOUND_SIZE,SND_FORMAT_LINEAR_16,
       SND_RATE_CODEC,soundStruct->channelCount," ");
  SNDConvertSound(soundStruct,&convertStruct);
  *temp = (short *)(convertStruct+convertStruct->dataLocation);
}


/******************** compress_sound() ********************************
 This function does most of the processing of the coding system.  It takes
 the sound signal, breaks it into windows, transforms the windows to the
 frequency domain, selects frequencies to represent the signal, encodes
 the frequency information, decodes the encoded data, transforms the
 signal back to the time domain.
 CALLED BY: main
 CALLS:  compute_slice_energy,compute_zero_crossings,
      compute_slope_changes,compute_low_freq_energy,noisy_value,
   zero_pad_frame,fft, freq_select,compute_noise_energy,quantize,
```

```
   expand,noise_frequencies,freq_to_time, error_correct
 INPUTS: signal - a pointer to an array of floats that represent the signal
 OUTPUTS: signal - a pointer to an array of floats that represent the signal
*************************************************************************/

void compress_sound(float *signal)
{float  txamp_vector[COUNT/2], txphase_vector[COUNT/2];
float   exp_amp_vector[COUNT/2], exp_phase_vector[COUNT/2];
float   time_vector[COUNT],compensate_vector[COUNT/2];
float   padded_frame[COUNT], amp_vector[COUNT], phase_vector[COUNT],
   real_vector[COUNT],imag_vector[COUNT];
float   timeframe_vector[SLICE_SIZE];
float   energy_vector[8],exp_energy_vector[8];
int     i,k, components,voiced,noisy_count,offset;
float   slice_energy,zero_crossings,low_freq_energy,slope_changes;
float   diff,previous;
FILE    *ftemp1,*ftemp2a,*ftemp2p,*ftemp3,*ftemp4,*fpcorrect,
      *fnoise_amp,*fnoise_phase;
frame_struct frame;




  if ((ftemp1 = fopen("selected.dat","w")) == NULL) {
     printf("\n*** Can't create %s ***","selected.dat");
     exit(0);
   }

  if ((ftemp2a = fopen("fourier_amp.dat","w")) == NULL) {
     printf("\n*** Can't create %s ***","fourier.dat");
     exit(0);
   }
  if ((ftemp2p = fopen("fourier_phase.dat","w")) == NULL) {
     printf("\n*** Can't create %s ***","fourier.dat");
     exit(0);
   }

  if ((ftemp3 = fopen("decide.dat","w")) == NULL) {
     printf("\n*** Can't create %s ***","decide.dat");
     exit(0);
   }
  if ((ftemp4 = fopen("train.dat","w")) == NULL) {
     printf("\n*** Can't create %s ***","decide.dat");
     exit(0);
   }
```

```
  if ((fnoise_amp = fopen("noise_amp.dat","r")) == NULL) {
     printf("\n*** Can't create %s ***","noise_amp.dat");
     exit(0);
   }

  if ((fnoise_phase = fopen("noise_phase.dat","r")) == NULL) {
     printf("\n*** Can't create %s ***","noise_phase.dat");
     exit(0);
   }

   previous = 1.0;


while (offset <  SOUND_SIZE)
 {
   slice_size = 160;

   if ((offset + 512) > SOUND_SIZE)
    { slice_size = SOUND_SIZE - offset;
   noisy_count = 1;
   loopk(slice_size) timeframe_vector[k] = signal[offset+k];
    }
    else
     {
      noisy_count = 0;
      loopi(5)
       {
        loopk(32)
         timeframe_vector[i*32+k] = signal[offset+(i*32)+k];
        slice_energy = compute_slice_energy(timeframe_vector,i);
        zero_crossings = compute_zero_crossings(timeframe_vector,i);
      slope_changes = compute_slope_changes(timeframe_vector,i);
       low_freq_energy = compute_low_freq_energy(timeframe_vector,i);
      noisy_count += noisy_value(slice_energy,zero_crossings,
                          slope_changes,low_freq_energy,
                    &previous);
     }

      i = 5;
      loopk(32)
       timeframe_vector[i*32+k] = signal[offset+(i*32)+k];
      slice_energy = compute_slice_energy(timeframe_vector,i);
    zero_crossings = compute_zero_crossings(timeframe_vector,i);
    slope_changes = compute_slope_changes(timeframe_vector,i);
    low_freq_energy = compute_low_freq_energy(timeframe_vector,i);
    if (noisy_count < 2)
```

```
    noisy_count = 0;
   else
    noisy_count = 1;


     while ((noisy_count == noisy_value(slice_energy,zero_crossings,
                          slope_changes,low_freq_energy,
                &previous))
                &&(slice_size < 512))
    {
       slice_size += 32;
     i++;
     loopk(32)
         timeframe_vector[i*32+k] = signal[offset+(i*32)+k];
       slice_energy = compute_slice_energy(timeframe_vector,i);
     zero_crossings = compute_zero_crossings(timeframe_vector,i);
       slope_changes = compute_slope_changes(timeframe_vector,i);
     low_freq_energy = compute_low_freq_energy(timeframe_vector,i);

    }
  }
printf("offset = %d slice_size = %d noisy = %d \n",
       offset,slice_size,noisy_count);

 if (noisy_count)
  components = slice_size / 20 - 5;
 else
  components = slice_size / 20 - 1;

 zero_pad_frame(timeframe_vector,padded_frame);

 fft(padded_frame,amp_vector, phase_vector);


if(!(noisy_count))
 loopk(COUNT/2)
   {
 fprintf(ftemp2a,"%12.9f\n",amp_vector[k]);
    fprintf(ftemp2p,"%f\n",phase_vector[k]);
    }

 freq_select(amp_vector, phase_vector, txamp_vector,
       txphase_vector, compensate_vector,
    components);


 loopk(COUNT/2)
```

```
    fprintf(ftemp1,"%f\n",txamp_vector[k]);


if (noisy_count)
 compute_noise_energy(amp_vector,txamp_vector,energy_vector);

frame = quantize(txamp_vector,txphase_vector,noisy_count,slice_size,

      energy_vector);



expand(frame,exp_amp_vector,exp_phase_vector,exp_energy_vector);



if (noisy_count)
  noise_frequencies(exp_energy_vector,exp_amp_vector,exp_phase_vector,
          fnoise_amp,fnoise_phase);

freq_to_time(exp_amp_vector, exp_phase_vector, real_vector, imag_vector,
        time_vector);

if (offset)
 { diff = signal[offset-8]-signal[offset-7];
  signal[offset-8] -= diff/3.0;
  signal[offset-7] += diff/3.0;
  diff = signal[offset-7]-signal[offset-6];
  signal[offset-7] -= diff/3.0;
  signal[offset-6] += diff/3.0;
  diff = signal[offset-6]-signal[offset-5];
  signal[offset-6] -= diff/3.0;
  signal[offset-5] += diff/3.0;
  diff = signal[offset-5]-signal[offset-4];
  signal[offset-5] -= diff/3.0;
  signal[offset-4] += diff/3.0;
  diff = signal[offset-4]-signal[offset-3];
  signal[offset-4] -= diff/3.0;
  signal[offset-3] += diff/3.0;
  diff = signal[offset-3]-signal[offset-2];
  signal[offset-3] -= diff/3.0;
  signal[offset-2] += diff/3.0;
  diff = signal[offset-2]-signal[offset-1];
  signal[offset-2] -= diff/3.0;
  signal[offset-1] += diff/3.0;
  diff = signal[offset-1]-time_vector[0];
  signal[offset-1] -= diff/3.0;
```

```
        time_vector[0] += diff/3.0;
        diff = time_vector[0]-time_vector[1];
        time_vector[0] -= diff/3.0;
        time_vector[1] += diff/3.0;
        diff = time_vector[1]-time_vector[2];
        time_vector[1] -= diff/3.0;
        time_vector[2] += diff/3.0;
        diff = time_vector[2]-time_vector[3];
        time_vector[2] -= diff/3.0;
        time_vector[3] += diff/3.0;
        diff = time_vector[3]-time_vector[4];
        time_vector[3] -= diff/3.0;
        time_vector[4] += diff/3.0;
        diff = time_vector[4]-time_vector[5];
        time_vector[4] -= diff/3.0;
        time_vector[5] += diff/3.0;
        diff = time_vector[5]-time_vector[6];
        time_vector[5] -= diff/3.0;
        time_vector[6] += diff/3.0;
        diff = time_vector[6]-time_vector[7];
        time_vector[6] -= diff/3.0;
        time_vector[7] += diff/3.0;
        diff = time_vector[7]-time_vector[8];
        time_vector[7] -= diff/3.0;
        time_vector[8] += diff/3.0;
      }


     for(k = 0;k < slice_size;k++)
    signal[offset+k] = time_vector[k];

offset += slice_size;

 } /** end while (offset < SOUND_SIZE) **/

 error_correct(signal);

    fclose(ftemp1);
    fclose(ftemp2a);
    fclose(ftemp2p);
    fclose(ftemp3);
    fclose(ftemp4);
    fclose(fnoise_amp);
    fclose(fnoise_phase);

 }       /** end compress_sound **/
```

```
/******************** output_sound ************************************
 This function writes a sound file.
 CALLED BY: main
 CALLS: none
 INPUTS: temp - a pointer to an array of short ints, this is where
            the 16 bit linear version of the signal is kept
         outfile - a string that specifies the output sound file name
 OUTPUTS: none
*****************************************************************/

void output_sound(short **temp,char *outfile)

{
  SNDAlloc(&newStruct,SOUND_SIZE,SND_FORMAT_MULAW_8,SND_RATE_CODEC,
          soundStruct→channelCount," ");
    newStruct→dataLocation = *temp;

    SNDConvertSound(convertStruct,&newStruct);

    SNDWriteSoundfile(outfile,convertStruct);
}




/************************** find_smallest ********************************
 This function picks the smallest element out of an array of floats.
 CALLED BY: pick_n_largest
 CALLS: none
 INPUTS: temp - an array of floats
         size - the length of temp
 OUTPUTS: smallest - the smallest float in the list
          smallest_index - the index of the smallest element in the
          array.
*****************************************************************/

void find_smallest(float *temp,float *smallest,

          int *smallest_index, int size)



{ int i;
```

```
  *smallest = temp[0];
  *smallest_index = 0;

  for( i = 1; i < size; i++)
    if( temp[i] < *smallest)
   { *smallest = temp[i];
    *smallest_index = i;
   }
}

/************************** pick_n_largest ******************************
 This function picks the n largest values from an array of floats.
 CALLED BY: freq_select
 CALLS:  find_smallest
 INPUTS: in - an array of floats
       n - the number of values to pick
       size - the length of in
 OUTPUTS: in - the same as teh original array, except the size - n
       smallest values are set to zero
************************************************************************/
void pick_n_largest(float *in,int n,int size)

{
 float temp[SLICE_SIZE], smallest;
 int  index[SLICE_SIZE],smallest_index,i;

 loopi(n)
```

```
    {temp[i] = in[i];
     index[i] = i;
    }

  find_smallest(temp,&smallest,&smallest_index,n);

  for( i = n; i < size; i++)
    if( in[i] > smallest )
    {temp[smallest_index] = in[i];
     index[smallest_index] = i;
     find_smallest(temp,&smallest,&smallest_index,n);
    }

  loopi(size)
     in[i] = 0.0;

  loopi(n)
     in[index[i]] = temp[i];
}
```

/********** *pick_n_largest1* **********
This function picks the n largest values from an array of floats.
CALLED BY: freq_select
CALLS: none

```
 INPUTS: in - an array of floats
       n - the number of values to pick
       size - the length of in
 OUTPUTS: in - the same as the original array, except the size - n
       smallest values are set to zero
*****************************************************************************/
void pick_n_largest1(float *in,float *out,int n)



{

 float temp[COUNT/2], largest;

 int   largest_index,i,j;


loopi(COUNT/2)
  { out[i] = 0.0;
   temp[i] = in[i];

  }

loopi(n)

 { largest = 0.0;
    loopj(COUNT/2)
      if (temp[j] > largest)
     { largest_index = j;
       largest = temp[j];
     }

    out[largest_index] = temp[largest_index];
    temp[largest_index] = 0.0;

    if (largest_index > 0)
      {out[largest_index] += temp[largest_index-1];
      temp[largest_index-1] = 0.0;
      }
    if (largest_index > 1)
      {out[largest_index] += temp[largest_index-2];
      temp[largest_index-2] = 0.0;
      }
    if (largest_index < COUNT/2-1)
      {out[largest_index] += temp[largest_index+1];
      temp[largest_index+1] = 0.0;
      }
```

```
    if (largest_index < COUNT/2 - 2)
      {out[largest_index] += temp[largest_index+2];
       temp[largest_index+2] = 0.0;
      }

  }


}
```

```
/******************** noise_frequencies ******************************
 This function creates a noiselike spectrum according to the energy in each of 16 frequency
bands. The spectrum of the noise should have its energy roughly distributed the same as
the original slice of speech.
 CALLED BY: compress_sound
 CALLS: none
 INPUTS: energy_vector - an array of floats
        txamp_vector - an array of floats representing the magnitudes of the transmitted
frequencies
        txphase_vector - an array of floats representing the phases of the transmitted
frequencies
 OUTPUTS: txamp_vector - an array of floats representing the magnitudes of the transmitted
frequencies with noise added
        txphase_vector - an array of floats representing the phases of the transmitted
frequencies with noise added
*************************************************************************/
noise_frequencies(float *energy_vector,float *txamp_vector,
            float *txphase_vector,FILE *fnoise_amp,FILE *fnoise_phase)
{ int i;
  float noise[COUNT/2];
  float noise_energy,comp_factor;

loopi(COUNT/2)
  {
   fscanf(fnoise_amp,"%f",&noise[i]);
   fscanf(fnoise_phase,"%f",&txphase_vector[i]);
  }

noise_energy = 0.0;
for (i = 0; i ≤ 14; i++)
   noise_energy += noise[i]*noise[i];

   comp_factor = sqrt(energy_vector[0]/noise_energy);
for (i = 0; i ≤ 14; i++)
```

```
  if(txamp_vector[i] == 0.0)
    txamp_vector[i] = comp_factor*noise[i];


noise_energy = 0.0;
for (i = 15; i ≤ 32; i++)
   noise_energy += noise[i]*noise[i];

   comp_factor = sqrt(energy_vector[1]/noise_energy);
for (i = 15; i ≤ 32; i++)
  if(txamp_vector[i] == 0.0)
   txamp_vector[i] = comp_factor*noise[i];


noise_energy = 0.0;
for (i = 33; i ≤ 53; i++)
   noise_energy += noise[i]*noise[i];

   comp_factor = sqrt(energy_vector[2]/noise_energy);
for (i = 33; i ≤ 53; i++)
  if(txamp_vector[i] == 0.0)
   txamp_vector[i] = comp_factor*noise[i];


noise_energy = 0.0;
for (i = 54; i ≤ 79; i++)
   noise_energy += noise[i]*noise[i];

   comp_factor = sqrt(energy_vector[3]/noise_energy);
for (i = 54; i ≤ 79; i++)
  if(txamp_vector[i] == 0.0)
   txamp_vector[i] = comp_factor*noise[i];


noise_energy = 0.0;
for (i = 80; i ≤ 111; i++)
   noise_energy += noise[i]*noise[i];

   comp_factor = sqrt(energy_vector[4]/noise_energy);
for (i = 80; i ≤ 111; i++)
  if(txamp_vector[i] == 0.0)
   txamp_vector[i] = comp_factor*noise[i];


noise_energy = 0.0;
for (i = 112; i ≤ 149; i++)
   noise_energy += noise[i]*noise[i];
```

```
   comp_factor = sqrt(energy_vector[5]/noise_energy);
for (i = 112; i ≤ 149; i++)
  if(txamp_vector[i] == 0.0)
   txamp_vector[i] = comp_factor*noise[i];


noise_energy = 0.0;
for (i = 150; i ≤ 197; i++)
   noise_energy += noise[i]*noise[i];

   comp_factor = sqrt(energy_vector[6]/noise_energy);
for (i = 150; i ≤ 197; i++)
  if(txamp_vector[i] == 0.0)
   txamp_vector[i] = comp_factor*noise[i];


noise_energy = 0.0;
for (i = 198; i ≤ 255; i++)
   noise_energy += noise[i]*noise[i];

   comp_factor = sqrt(energy_vector[7]/noise_energy);
for (i = 198; i ≤ 255; i++)
  if(txamp_vector[i] == 0.0)
   txamp_vector[i] = comp_factor*noise[i];



}       /* end function noise_frequencies */

/******************** compute_noise_energy ******************************
 This function computes the energy in each of 16 mel-scaled frequency bands.
 CALLED BY: compress_sound
 CALLS:  none
 INPUTS: amp_vector - an array of floats representing the magnitudes of the original
spectrum
        txamp_vector - an array of floats representing the magnitudes of the transmitted
frequencies
 OUTPUTS: energy_vector - an array of floats representing the energy of the noisy spectrum
minus the energy of the transmitted frequencies
*************************************************************************/
compute_noise_energy(float *amp_vector,float *txamp_vector,
        float *energy_vector)
{  int i;

 loopi(16)
  energy_vector[i] =  0.0;
```

```
for (i = 0; i ≤ 7; i++)
  if(txamp_vector[i] == 0.0)
    energy_vector[0] += amp_vector[i]*amp_vector[i];

for (i = 8; i ≤ 14; i++)
  if(txamp_vector[i] == 0.0)
   energy_vector[0] += amp_vector[i]*amp_vector[i];

for (i = 15; i ≤ 22; i++)
  if(txamp_vector[i] == 0.0)
   energy_vector[1] += amp_vector[i]*amp_vector[i];

for (i = 23; i ≤ 32; i++)
  if(txamp_vector[i] == 0.0)
   energy_vector[1] += amp_vector[i]*amp_vector[i];

for (i = 33; i ≤ 42; i++)
  if(txamp_vector[i] == 0.0)
   energy_vector[2] += amp_vector[i]*amp_vector[i];

for (i = 43; i ≤ 53; i++)
  if(txamp_vector[i] == 0.0)
   energy_vector[2] += amp_vector[i]*amp_vector[i];

for (i = 54; i ≤ 65; i++)
  if(txamp_vector[i] == 0.0)
   energy_vector[3] += amp_vector[i]*amp_vector[i];

for (i = 66; i ≤ 79; i++)
  if(txamp_vector[i] == 0.0)
   energy_vector[3] += amp_vector[i]*amp_vector[i];

for (i = 80; i ≤ 94; i++)
  if(txamp_vector[i] == 0.0)
   energy_vector[4] += amp_vector[i]*amp_vector[i];

for (i = 95; i ≤ 111; i++)
  if(txamp_vector[i] == 0.0)
   energy_vector[4] += amp_vector[i]*amp_vector[i];

for (i = 112; i ≤ 129; i++)
  if(txamp_vector[i] == 0.0)
   energy_vector[5] += amp_vector[i]*amp_vector[i];

for (i = 130; i ≤ 149; i++)
  if(txamp_vector[i] == 0.0)
```

```
   energy_vector[5] += amp_vector[i]*amp_vector[i];

for (i = 150; i ≤ 172; i++)
  if(txamp_vector[i] == 0.0)
   energy_vector[6] += amp_vector[i]*amp_vector[i];

for (i = 173; i ≤ 197; i++)
  if(txamp_vector[i] == 0.0)
   energy_vector[6] += amp_vector[i]*amp_vector[i];

for (i = 198; i ≤ 225; i++)
  if(txamp_vector[i] == 0.0)
   energy_vector[7] += amp_vector[i]*amp_vector[i];

for (i = 226; i < 255; i++)
  if(txamp_vector[i] == 0.0)
   energy_vector[7] += amp_vector[i]*amp_vector[i];

 loopi(16)
  energy_vector[i] *= 0.25;

}       /* end function simulate_noise */

/******************** compute_slice_energy ******************************
 This function computes the energy in a signal.
 CALLED BY: compress_sound
 CALLS:  none
 INPUTS: signal - an array of floats which contains the signal
       offset - an integer offset into the signal array
 OUTPUTS: energy  - the energy in the signal
************************************************************************/
float compute_slice_energy(float *slice,int offset)

{
 float energy = 0.0;
 int i;

  loopi(32) energy += slice[offset*32+i]*slice[offset*32+i];

  return energy;
 }

/******************** compute_zero_crossings ****************************
 This function computes the number of zero crossings in a portion of a signal.
 CALLED BY: compress_sound
 CALLS:  none
 INPUTS: signal - an array which contains the signal
```

```
      offset - an integer offset into the signal array
 OUTPUTS: zero_crossings - the number of zero crossings divided by the length of the
slice
*****************************************************************************/
float compute_zero_crossings(float *slice,int offset)
{
 float zero_crossings = 0.0;
 int i;

    loopi(32)
     {
  if(i > 1)
    if(((slice[offset*32+i-1] > 0)&&(slice[offset*32+i] < 0))||
       ((slice[offset*32+i-1] < 0)&&(slice[offset*32+i] > 0)))

     zero_crossings += 1.0;
     }


  zero_crossings /= 32;
  return zero_crossings;

}

/******************** compute_slope_changes ******************************
 This function computes the number of slope sign changes in a portion of a signal.
 CALLED BY: compress_sound
 CALLS:  none
 INPUTS: signal - an array which contains the signal
      offset - an integer offset into the signal array
 OUTPUTS: slope_changes - the number of slope sign changes divided by the length of
the slice
*****************************************************************************/
float compute_slope_changes(float *slice,int offset)
{ int i;
  float slope_changes = 0.0;

  loopi(32)
    if ((i > 0)&&(i<31))
    if(((slice[offset+i] > slice[offset+i-1])&&
       (slice[offset+i] > slice[offset+i+1]))||
          ((slice[offset+i] < slice[offset+i-1])&&
       (slice[offset+i] < slice[offset+i+1])))
      slope_changes += 1.0;
  return slope_changes/32.0;

}
```

```
/******************** compute_low_freq_energy ******************************
 This function computes the low frequency energy ratio in a portion of a signal.
 CALLED BY: compress_sound
 CALLS: none
 INPUTS: signal - an array which contains the signal
       offset - an integer offset into the signal array
 OUTPUTS: low_freq_energy - the ratio of energy below 500 Hz to the total energy for a
portion of a signal
*****************************************************************************/
float compute_low_freq_energy(float *slice,int offset)

{
 float low_freq_energy = 0.0;
 float padded_slice[COUNT],amp_vector[COUNT],phase_vector[COUNT];
 int i;
 float energy = 0.0;


for (i = 32; i < COUNT; i++)
  padded_slice[i] = 0.0;

loopi(32)
 padded_slice[i] = slice[offset*32+i];

  fft(padded_slice,amp_vector,phase_vector);

  for (i = 1; i ≤ 32; i++)

   low_freq_energy += amp_vector[i]*amp_vector[i];

  energy = low_freq_energy;
  for (i = 33; i < 255; i++)
    energy += amp_vector[i]*amp_vector[i];

  return low_freq_energy/energy;
 }


/******************** noisy_value ***********************************
 This function returns the results of the noiselike/periodic decision.  Inputs are feed into
a recurrent neural network in order to make the decision. The decision is made based on
the input values plus the weights that were obtained through training the recurrent net.
 CALLED BY: compress_sound
 CALLS: none
 INPUTS: energy - the total energy in a portion of the signal
       zero_crossings - the number of zero crossings in a portion of the signal
```

```
      slope_changes - the number of sign slope changes in a portion of the signal
       low_frequency_energy - the ratio of energy below 500 Hz to the total energy for a
portion of a signal
   previous - the previous output of the recurrent neural net
 OUTPUTS: previous - the new output of the recurrent neural net, used for the next noisy
value calculation
       noisy_value - the resulting decision from the recurrent net
*****************************************************************/
int noisy_value(float energy,float zero_crossings,float slope_changes,
          float low_freq_energy,float *previous)
{ int noisy = 0;
  float node0,sum0;
  float w0,w1,w2,w3,w4,w5,w6;

w0 = -1.836152;
w1 = 56.509041;
w2 = -18.466383;
w3 = -2.888618;
w4 = 2.411863;
w5 = 3.701866;

  sum0 = w0 +
       w1 * energy +
       w2 * zero_crossings +
    w3 * slope_changes +
    w4 * low_freq_energy +
    w5 * *previous;

  node0 = 1/(1+exp(-sum0));

  if (node0 < 0.5 )
   noisy  = 1;
  *previous = node0;

  return noisy;

}


/******************** quantize ***********************************
 This function quantizes the frequency, amplitude, phase, and noise energy  information
for a given slice of speech.
 CALLED BY: compress_sound
 CALLS:  none
 INPUTS: trans_mag_vector - an array of magnitudes representing a portion of speech
       trans_phase_vector - an array of phases representing a portion of speech
    noisy - the result of the noiselike/periodic decision
```

```
   slice_size - the number of samples in the given portion of speech
   energy - an array representing the mel-scaled noise energy information
 OUTPUTS: frame - resulting frame of transmitted spectral and energy information
 ************************************************************************/
frame_struct quantize(float *trans_mag_vector,
                      float *trans_phase_vector,
                      int noisy,
                      int slice_size,
                      float *energy)

{ int i,components_to_send,components_sent,diff;
 float max,harmonic_energy = 0.0;
 frame_struct frame;

 frame.length = slice_size/40 - 5;
 frame.noisy = noisy;

 if (noisy)
  components_to_send = 4 + 2*frame.length;
 else
  components_to_send = 8 + 2*frame.length;

 max = 0.0;

 loopi(COUNT/2)
  if (trans_mag_vector[i] > max)
```

```
    max = trans_mag_vector[i];

frame.largest_magnitude = (int) (max*128.0*8.0);

i = 0;

while((trans_mag_vector[i] == 0.0)&& (i < 64))
 i++;

frame.base_frequency = i;

frame.base_magnitude = (int) (trans_mag_vector[frame.base_frequency] *
 8.0/max);
frame.base_phase = (int) ((trans_phase_vector[frame.base_frequency] + PI)
     * 8.0/PI);

diff = 1;
i++;
components_sent = 0;

while((i < COUNT/2)&&(components_sent ≤ components_to_send))
 {
  if((trans_mag_vector[i] ≠ 0.0)||(diff == 31))
```

```
    { frame.frequency_array[components_sent].offset = diff;
      frame.frequency_array[components_sent].magnitude =
        (int) (trans_mag_vector[i] * 8.0/max);
      frame.frequency_array[components_sent].phase =
        (int) ((trans_phase_vector[i] + PI) * 8.0/PI);
      harmonic_energy += trans_mag_vector[i]*trans_mag_vector[i];
      components_sent++;
      diff = 0;
     }
   i++;
   diff++;
 }

if (noisy)
 loopi(8)
  {
   if (energy[i]/harmonic_energy < 1.0)
    frame.energy[i] = (int) (energy[i]/harmonic_energy*64.0);
   else
    { frame.energy[i] = ((int) (energy[i]/harmonic_energy*4.0)) + 16;

     if (frame.energy[i] > 31)
      frame.energy[i] = 31;
    }
```

```
    }

    return frame;

}

/******************** expand ***********************************
 This function expands the transmitted frame for a given portion of speech.  This
function reverses the process of quantize. Frequency, amplitude, phase, and noise energy
information for a given slice of speech are extracted from a frame.
 CALLED BY: compress_sound
 CALLS:  none
 INPUTS: frame - frame of transmitted spectral and energy information
 OUTPUTS: exp_mag_vector - an array of magnitudes representing a portion of speech
      exp_phase_vector - an array of phases representing a portion of speech
      exp_energy_vector - an array representing the mel-scaled noise energy information
 ****************************************************************/
expand(frame_struct frame,

    float *exp_mag_vector,

    float *exp_phase_vector,

    float *exp_energy_vector)


{ int i,k,components_sent,noisy;

 float largest_magnitude,harmonic_energy = 0.0;


 loopi(COUNT/2)

  exp_mag_vector[i] = exp_phase_vector[i] = 0.0;
```

```
largest_magnitude = (float)frame.largest_magnitude/(128.0*8.0);
noisy = frame.noisy;

i = frame.base_frequency;

exp_mag_vector[i] = largest_magnitude * (frame.base_magnitude+0.5) / 8.0;
exp_phase_vector[i] = PI / 8.0 * frame.base_phase - PI;
harmonic_energy += exp_mag_vector[i]*exp_mag_vector[i];

if (noisy)
 components_sent = 4 + 2*frame.length;
else
 components_sent = 8 + 2*frame.length;


loopk(components_sent)
 {
  i+= frame.frequency_array[k].offset;
  exp_mag_vector[i] = largest_magnitude *
     (frame.frequency_array[k].magnitude+0.5) / 8.0;
  exp_phase_vector[i] = PI / 8.0 * frame.frequency_array[k].phase - PI;
  harmonic_energy += exp_mag_vector[i]*exp_mag_vector[i];
 }
```

```
 if (noisy)

   loopi(8)

     if (frame.energy[i] ≤ 15)

exp_energy_vector[i] = harmonic_energy * frame.energy[i] /64.0;

     else

exp_energy_vector[i] = harmonic_energy *

        (frame.energy[i] − 16)/4.0;

}


/******************** error_correct ************************************
This function attempts to correct the errors that were introduced in the process of com-
pressing the sound.  The function implements the recurrent neural network with weights
that were obtained through training.
 CALLED BY: compress_sound
 CALLS:  none
 INPUTS: signal - an array representing the reconstructed signal
 OUTPUTS: signal - an array representing the reconstructed signal modified
*****************************************************************************/
 void error_correct(float *signal)
{
  float w0,w1,w2,w3,w4,previous = 0.0 ;
  float *new_signal;
int i;


w0 = −0.001630;
w1 = 1.173535;
w2 = −0.283277;

for (i = 1; i < SOUND_SIZE; i++)
  { previous =  w0 +
            w1 * signal[i] +
       w2 * previous ;
   signal[i] = previous;
   }

}
```

```
/******************** zero_pad_frame ***********************************
This function zero pads a sequence for fft processing.
 CALLED BY: compress_sound
 CALLS: none
 INPUTS: timeframe_vector - an array representing a portion of the signal
 OUTPUTS: padded_frame - an array representing the zero padded version of the time-
frame_vector*****************************************************************/
zero_pad_frame (float *timeframe_vector,float  *padded_frame)

{
 int i;

for (i = slice_size; i < COUNT; i++)
   padded_frame[i] = 0;

for (i = 0; i < slice_size; i++)
   padded_frame[i] = timeframe_vector[i];
} /** end zero_pad_frame **/


/******************** fft ***********************************
This function the 512 point Fourier transform on a portion of the signal.
 CALLED BY: compress_sound, compute_low_freq_energy
 CALLS:  DSPAPWriteFloatArray,DSPAPfftr2a,DSPSetDMAReadMReg,
      DSPAPReadFloatArray,DSPSetDMAReadMReg,DSPAPFree
 INPUTS: padded_frame - an array representing the zero padded version of the time-
frame_vector
 OUTPUTS: amp_vector - an array of floats representing the amplitude spectrum for a
given portion of speech
      phase_vector - an array of floats representing the phase spectrum for a given portion
of speech
*****************************************************************/


fft(float *padded_frame,float *amp_vector,float *phase_vector)



{ float *sinTab = DSPAPSinTable(COUNT);
 float *cosTab = DSPAPCosTable(COUNT);
 float real_vector[COUNT],imag_vector[COUNT];
 float imag_frame[COUNT];
 int i;
```

```
DSPAPInit();
loopi(COUNT) {real_vector[i] = padded_frame[i]/COUNT;
        imag_vector[i] = 0.0; }

DSPAPWriteFloatArray(real_vector, REAL_DATA, 1, COUNT);
DSPAPWriteFloatArray(imag_vector, IMAG_DATA, 1, COUNT);

DSPAPWriteFloatArray(cosTab, COS_TABLE, 1, COUNT/2);
DSPAPWriteFloatArray(sinTab, SIN_TABLE, 1, COUNT/2);

 DSPAPfftr2a(COUNT,DATA_ADR,LOOKUP_ADR);


DSPSetDMAReadMReg(0);
DSPAPReadFloatArray(real_vector, REAL_DATA, COUNT/2, COUNT);
DSPAPReadFloatArray(imag_vector, IMAG_DATA, COUNT/2, COUNT);
DSPSetDMAReadMReg(-1);


DSPAPFree();
loopi(COUNT/2)
  { amp_vector[i]  = (float)sqrt( real_vector[i]*real_vector[i] +
                       imag_vector[i]*imag_vector[i]);

    phase_vector[i] = (float)atan2(imag_vector[i],real_vector[i]);
  }

}


/******************** ifft ************************************
This function the 512 point inverse Fourier transform on a portion of the signal.
 CALLED BY: freq_to_time
 CALLS:  DSPAPWriteFloatArray,DSPAPfftr2a,DSPSetDMAReadMReg,
     DSPAPReadFloatArray,DSPSetDMAReadMReg,DSPAPFree
 INPUTS: real_data - an array of floats representing the real portion of the complex spectrum
for a given portion of speech
        real_data - an array of floats representing the imaginary portion of the complex
spectrum for a given portion of speech
 OUPUTS: time_vector - an array representing the reconstructed version of the given
portion of speech
*****************************************************************/
ifft(float *real_data,float *imag_data,float *time_vector)
```

```
{ float *sinTab = DSPAPSinTable(COUNT);
 float *cosTab = DSPAPCosTable(COUNT);

 float imag_frame[COUNT];
 int i;
  loopi(COUNT/2) sinTab[i] = -sinTab[i];

 DSPAPInit();

 DSPAPWriteFloatArray(real_data, REAL_DATA, 1, COUNT);
 DSPAPWriteFloatArray(imag_data, IMAG_DATA, 1, COUNT);

 DSPAPWriteFloatArray(cosTab, COS_TABLE, 1, COUNT/2);
 DSPAPWriteFloatArray(sinTab, SIN_TABLE, 1, COUNT/2);
 DSPAPfftr2a(COUNT,DATA_ADR,LOOKUP_ADR);


 DSPSetDMAReadMReg(0);
 DSPAPReadFloatArray(time_vector, REAL_DATA, COUNT/2, COUNT);
 DSPSetDMAReadMReg(-1);


 DSPAPFree();
}



/******************** freq_select ***********************************
This function chooses which frequencies to represent a given portion of speech.
 CALLED BY: compress_sound
 CALLS:  pick_n_largest
 INPUTS: amp_vector - an array of floats representing the amplitude spectrum for a given
portion of speech
      phase_vector - an array of floats representing the phase spectrum for a given portion
of speech
   components - the integer number of frequencies to be selected
 OUPUTS: txamp_vector - an array of floats representing the selected amplitude spectrum
for a given portion of speech
       txphase_vector - an array of floats representing the selected phase spectrum for a
given portion of speech
*********************************** ********************************/
freq_select (float *amp_vector, float *phase_vector,
      float *txamp_vector, float *txphase_vector,
      int components)


{
```

```
float  glottal_amp, harm_amp, decide_vector[COUNT/2];
int    n;
int    glottal_bin, glot_harm, harm_bin,i,x,y;



 loopi(COUNT/2)
 {  decide_vector[i] = amp_vector[i];
    txamp_vector[i] = txphase_vector[i] = 0.0 ;
 }



 pick_n_largest(decide_vector,components,COUNT/2);

 loopi(COUNT/2)
    if (decide_vector[i] ≠ 0.0)
      { txamp_vector[i] = amp_vector[i];
     txphase_vector[i] = phase_vector[i];
     }

 if (decide_vector[0] ≠ 0.0)
      { txamp_vector[0] = amp_vector[0];
    txphase_vector[0] = phase_vector[0];
    txphase_vector[1] = phase_vector[1];
    txamp_vector[1] = .4*amp_vector[0];
    }

 if (decide_vector[COUNT/2-1] ≠ 0.0)
      { txamp_vector[COUNT/2-1] = amp_vector[COUNT/2-1];
    txphase_vector[COUNT/2-1] = phase_vector[COUNT/2-1];
    txphase_vector[COUNT/2-2] = phase_vector[COUNT/2-2];
    txamp_vector[COUNT/2-2] = .4*amp_vector[COUNT/2-1];
    }


for(i = 1; i < COUNT/2 - 1; i++)
  if (decide_vector[i] ≠ 0.0)
      { txamp_vector[i] = amp_vector[i];
    txamp_vector[i-1] = .4*amp_vector[i];
    txamp_vector[i+1] = .4*amp_vector[i];
    txphase_vector[i] = phase_vector[i];
    txphase_vector[i-1] = phase_vector[i-1];
    txphase_vector[i+1] = phase_vector[i+1];
   }

for(i = 2; i < COUNT/2 - 2; i++)
```

```
  if((txamp_vector[i-1] > txamp_vector[i])&&
    (txamp_vector[i+1] > txamp_vector[i]))
    txamp_vector[i] = (txamp_vector[i-1]+txamp_vector[i+1])/2.4;

} /* end freq_select */



/******************** freq_to_time ***********************************
This function converts the frequency domain version of a portion of speech into a time
domain version of speech.
 CALLED BY: compress_sound
 CALLS: ifft
 INPUTS: amp_vector - an array of floats representing the amplitude spectrum for a given
portion of speech
      phase_vector - an array of floats representing the phase spectrum for a given portion
of speech
 OUPUTS: time_vector - an array representing the time domain version of a portion of
speech
*************************************************************************/

freq_to_time(float *amp_vector,float *phase_vector,float *real_vector,
        float *imag_vector,float *time_vector)


{
int i;


 real_vector[0] = amp_vector[0];
 imag_vector[0] = 0.0;

 loopi(COUNT/2)
  { real_vector[i] = real_vector[COUNT - i ]
            = amp_vector[i]*cos(phase_vector[i]);
    imag_vector[i] = amp_vector[i]*sin(phase_vector[i]);
    imag_vector[COUNT - i ] = -imag_vector[i];

  }


ifft(real_vector,imag_vector,time_vector);

}
```

```
main (int argc, char **argv)
{     int m;
   short *temp; /* short int representation of original data */
   float *signal;
   float max;
   FILE *f1,*f2;

   mySound=[Sound new];


   get_sound(&temp,argv[1]);  /* get signal, temp has integer representation of signal */

signal = calloc(SOUND_SIZE,sizeof(float));

   loopm(SOUND_SIZE)
      signal[m]=(float) temp[m];
      max = 0.0;
      loopm(SOUND_SIZE)
       if (max < abs(signal[m]))
         max = abs(signal[m]);

      loopm(SOUND_SIZE)    /* peak normalize and convert signal to float */
       signal[m] /= max;



   if ((f1 = fopen("signal.dat","w")) == NULL) {
      printf("\n*** Can't create %s ***","signal.dat");
      exit(0);
   }
   if ((f2 = fopen("signal_int.dat","w")) == NULL) {
      printf("\n*** Can't create %s ***","signal_int.dat");
      exit(0);
   }

   loopm(SOUND_SIZE)
       fprintf(f1,"%f\n",signal[m]);

   fclose(f1);

   loopm(SOUND_SIZE)
       fprintf(f2,"%d\n",temp[m]);

   fclose(f2);


   printf("Got data successfully!\n");
```

```
    compress_sound(signal);    /* process the signal */



    if ((f1 = fopen("signal2.dat","w")) == NULL) {
      printf("\n*** Can't create %s ***","signal2.dat");
      exit(0);
    }

    loopm(SOUND_SIZE)  /* store float representation of reconstruction */
        fprintf(f1,"%f\n",signal[m]);

    fclose(f1);

      loopm(SOUND_SIZE)
       signal[m] *= max;


    loopm(SOUND_SIZE)
     temp[m] = (short) signal[m];


    free(signal);

    output_sound(&temp,argv[2]);  /* store sound to sound file */

}
```

Bibliography

1 Alenquer, Luis M. F. *Rule Based Sinusoidal Encoding of Speech.* MS thesis, AFIT/GE/ENG/90M-1, Air Force Institute of Technology, Wright-Patterson AFB OH, March 1990.

2. Bashir, Nadeem A. *Phoneme Adjustment in Enhanced Speech.* MS thesis, AFIT/ENG/89M-2, Air Force Institute of Technology, Wright-Patterson AFB OH, March 1989 (AD–A206 357).

3. Daubechies, Ingrid. "The wavelet transform, time-frequency localization, and signal analysis," *IEEE Transactions on Information Theory, 36*(5):961–1005 (September 1990).

4. Fulton, Doug. *NeXT Sound, Music, and Signal Processing Concepts.* NeXT Computer Inc., 900 Chesapeake Drive, Redwood City, CA 94063, 1990.

5. Goble, James R. *Face Recognition Using the Discrete Cosine Transform.* MS thesis, AFIT/GEO/ENG/91D, Air Force Institute of Technology, Wright-Patterson AFB OH, December 1991.

6. Gold, Bernard and Charles M. Rader. "Systems for Compressing the Bandwidth of Speech," *IEEE Transactions on Audio and Electroacoustics, AU-15*(3):131–136 (September 1967).

7. Guyton, Arthur C. *Textbook of Medical Physiology*, chapter The Sense of Hearing. Philadelphia: W.B. Saunders, 1991.

8. Kabrisky, Matthew. "Personal Conversation." March 1991.

9. Kabrisky, Matthew, et al. AFIT Application of Wavelets – Biological Connections, Presentation to the Applications of Wavelets to Signal Processing Conference, Air Force Institute of Technology, Wright Patterson AFB OH, 22 March 1991.

10. Ladefoged, Peter. "The Phonetic Basis for Computer Speech Processing." In Fallside, Frank and William A. Woods, editors, *Computer Speech Processing*, chapter 1, London: Prentice-Hall International, 1985.

11. Lindsey, Randall L. *Function Prediction Using Recurrent Neural Networks.* MS thesis, AFIT/GEO/ENG/91D-16, Air Force Institute of Technology, Wright-Patterson AFB OH, December 1991.

12. Lindsey, Randall L. "Personal Conversation." August 1991.

13. Lippmann, Richard P. "Review of Neural Networks for Speech Recognition," *Neural Computation, 1*:1–38 (1989).

14. McMillan, Vance M. *Rule-Based Frequency Domain Speech Coding.* MS thesis, AFIT/GE/ENG/90D, Air Force Institute of Technology, Wright-Patterson AFB OH, December 1990.

15. Mendenhall, William, et al. *Mathematical Statistics with Applications*. Boston MS: PWS-KENT Publishing Company, 1990.

16. Mish, Frederick C., editor. *Webster's Ninth New Collegiate Dictionary*. Merriam Webster Inc., 1990.

17. Papamichalis, Panos E. *Practical Approaches to Speech Coding*. Englewood Cliffs NJ: Prentice-Hall, Inc., 1987.

18. Parsons, Thomas W. *Voice and Speech Processing*. New York: McGraw-Hill Book Company, 1987.

19. Quatieri, Thomas E. and R.J. McAulay. "Speech transformation based on a sinusoidal representation." In *Proceedings of the International Conference of Acoustics, Speech, and Signal Processing*, pages 13.5.1–13.5.4, 1985.

20. Quatieri, Thomas E. and R.J. McAulay. "Sine-Wave Phase Coding at Low Data Rates." In *Proceedings of the International Conference of Acoustics, Speech, and Signal Processing*, pages 577–580, 1991.

21. Rabiner, Lawrence R. and Ronald W. Schafer. *Digital Signal Processing of Speech*. Englewood Cliffs NJ: Prentice-Hall, Inc., 1978.

22. Ricart, Richard. *Speech Coding and Compression Using Wavelets and Lateral Inhibitory Networks*. MS thesis, AFIT/GE/ENG/90D, Air Force Institute of Technology, Wright-Patterson AFB OH, December 1990.

23. Rogers, Steven K. "Personal Conversation." July 1991.

24. Rogers, Steven K. and Matthew Kabrisky. *An Introduction to Biological and Artificial Neural Networks for Pattern Recognition*. Washington: SPIE Optical Engineering Press, 1991.

25. Ruck, Dennis W. *Characterization of Multilayer Perceptrons and their Application to Multisensor Automatic Target Detection*. PhD dissertation, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1990 (AFIT/DS/ENG/90-2).

26. Ruck, Dennis W., et al. "The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant," *IEEE Transactions on Neural Networks*, *1*(4) (1990).

27. Smiley, Steven E. *Image Segmentation Using Affine Wavelets*. MS thesis, AFIT/EN/ENG/91-D, Air Force Institute of Technology, Wright-Patterson AFB OH, December 1991.

28. Stowe, Francis Scott. *Analysis of an Automated Training Strategy for a Speech Recognition System Which Uses a Kohonen Neural Network, Dynamic Programming, and Multi-Feature Fusion*. MS thesis, AFIT/GE/ENG/90D, Air Force Institute of Technology, Wright-Patterson AFB OH, December 1990. Research in Progress.

29. Suarez, Pedro F. *Face Recognition*. MS thesis, AFIT/GEO/ENG/91D, Air Force Institute of Technology, Wright-Patterson AFB OH, December 1991.

30. Sundberg, Johan. "The Acoustics of the Singing Voice," *Scientific American*, pages 82–91+ (March 1977).

31. Tarr, Gregory L. *Neural Networks for Image Segmentation in Cluttered Data.* PhD dissertation, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1991 (AFIT/DS/ENG/91).